REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1995	Master's Thesis

4. TITLE AND SUBTITLE
NONLINEAR TIME SERIES ANALYSIS

5. FUNDING NUMBERS

6. AUTHOR(S)
James A. Stewart, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOA/ENG/95M-01

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
James R. Stright, Capt, USAF, WL/MNGA
Bldg 13 Ste 206, 101W. Eglin Blvd
Eglin AFB, FL 32542-6810

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis applies neural network feature selection techniques to multivariate time series data to improve prediction of a target time series. Two approaches to feature selection are used. First, a subset enumeration method is used to determine which financial indicators are most useful for aiding in prediction of the S&P 500 futures daily price. The candidate indicators evaluated include RSI, Stochastics and several moving averages. Results indicate that the Stochastics and RSI indicators result in better prediction results than the moving averages. The second approach to feature selection is calculation of individual saliency metrics. A new decision boundary based individual saliency metric, and a classifier independent saliency metric are developed and tested. Ruck's saliency metric, the decision boundary based saliency metric, and the classifier independent saliency metric are compared for a data set consisting of the RSI and Stochastics indicators as well as delayed closing price values. The decision based metric and the Ruck metric results are similar, but the classifier independent metric agrees with neither of the other metrics. The nine most salient features, determined by the decision boundary based metric, are used to train a neural network and the results are presented and compared to other published results.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Neural Networks, Time Series Prediction, Feature Selection	89
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

AFIT/GOA/ENG/95M-01

NONLINEAR TIME SERIES ANALYSIS

THESIS

James A. Stewart
Captain, USAF

AFIT/GOA/ENG/95M-01

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1995	Master's Thesis

4. TITLE AND SUBTITLE
NONLINEAR TIME SERIES ANALYSIS

5. FUNDING NUMBERS

6. AUTHOR(S)
James A. Stewart, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOA/ENG/95M-01

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
James R. Stright, Capt, USAF, WL/MNGA
Bldg 13 Ste 206, 101W. Eglin Blvd
Eglin AFB, FL 32542-6810

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis applies neural network feature selection techniques to multivariate time series data to improve prediction of a target time series. Two approaches to feature selection are used. First, a subset enumeration method is used to determine which financial indicators are most useful for aiding in prediction of the S&P 500 futures daily price. The candidate indicators evaluated include RSI, Stochastics and several moving averages. Results indicate that the Stochastics and RSI indicators result in better prediction results than the moving averages. The second approach to feature selection is calculation of individual saliency metrics. A new decision boundary based individual saliency metric, and a classifier independent saliency metric are developed and tested. Ruck's saliency metric, the decision boundary based saliency metric, and the classifier independent saliency metric are compared for a data set consisting of the RSI and Stochastics indicators as well as delayed closing price values. The decision based metric and the Ruck metric results are similar, but the classifier independent metric agrees with neither of the other metrics. The nine most salient features, determined by the decision boundary based metric, are used to train a neural network and the results are presented and compared to other published results.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Neural Networks, Time Series Prediction, Feature Selection	89
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

AFIT/GOA/ENG/95M-01

NONLINEAR TIME SERIES ANALYSIS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Operations Research

James A. Stewart, B.S.A.E.

Captain, USAF

March, 1995

Approved for public release; distribution unlimited

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

Acknowledgements

For their technical expertise and encouragement, I would like to thank the Friday afternoon stock market group. In particular, Dr. Steve Rogers, the founder of the stock market group, and my thesis advisor, was a constant source of encouragement and intellectual stimulation. He made a somewhat dreary task both enjoyable and educational. Dr. Dennis Quinn, also a member of the stock market group, is responsible for turning the writing herein into a somewhat intelligible form through his thorough reading of and feedback on this thesis.

Finally, I would like to thank my wife, Janis who often felt like a single parent during this work, and my four children, Amanda, Tristan, Elizabeth and Trevor for putting up with my obsession for completing this project, often at their expense.

James A. Stewart

Table of Contents

List of Figures

List of Tables

Abstract

This thesis applies neural network feature selection techniques to multivariate time series data to improve prediction of a target time series. Two approaches to feature selection are used. First, a subset enumeration method is used to determine which financial indicators are most useful for aiding in prediction of the S&P 500 futures daily price. The candidate indicators evaluated include RSI, Stochastics and several moving averages. Results indicate that the Stochastics and RSI indicators result in better prediction results than the moving averages. The second approach to feature selection is calculation of individual saliency metrics. A new decision boundary based individual saliency metric, and a classifier independent saliency metric are developed and tested. Ruck's saliency metric, the decision boundary based saliency metric, and the classifier independent saliency metric are compared for a data set consisting of the RSI and Stochastics indicators as well as delayed closing price values. The decision based metric and the Ruck metric results are similar, but the classifier independent metric agrees with neither of the other metrics. The nine most salient features, determined by the decision boundary based metric, are used to train a neural network and the results are presented and compared to other published results.

NONLINEAR TIME SERIES ANALYSIS

I. Introduction

1.1 Background

Accurately predicting future values of a time series has many benefits. Predicting the future position of an aircraft allows more accurate tracking. Accurate prediction of the motion of a pilot's head motion in a cockpit environment allows more efficient updating of helmet mounted displays [19]. Both aircraft position and head motion are examples of time series that are difficult to predict, because they are the result of nonlinear processes. Two contemporary prediction methods for nonlinear processes are state space reconstruction and neural networks.

The state space reconstruction method attempts to reconstruct the nonlinear dynamic system which produces the time series. This method is based on the theory of embedology and its generalization [8, 29]. Stright explained and applied state space reconstruction to time series prediction in his dissertation [29]. Sauer enhanced the basic state space construction method and successfully applied it to a nonlinear time series [26].

Neural networks are also used to model the nonlinear driving function. Both state space reconstruction and neural networks use past samples of the target time series to predict future values of the series. Waibel developed his time-delay neural (TDNN) architecture specifically for time series applications [34]. Using Waibel's TDNN, Wan successfully predicted a nonlinear time series [35, 11, 10].

Past samples of a time series are not always the only information available. Often, other time series are related in some way to the target time series. The price of a stock, for example, is related to the trading volume of that stock. These other helper time series can be either derived from the target time series, or totally independent. Stock market indicator data, for instance, is almost completely derived from the target price data. Using the Dow Jones

index to help predict the S&P 500 futures price is an example of a more or less independent helper time series. Neither state space reconstruction, nor neural network methods of time series prediction incorporate useful information from these other related time series.

Techniques using multiple time series to predict a target time series require a means of determining the relevance or *saliency* of a particular time series to successful prediction of a target time series. Ruck, Priddy and Tarr came up with derivative based saliency metrics for neural networks [24, 27]. Steppe used these saliency metrics to implement a backward stepwise feature elimination solution to this saliency problem for neural networks [27]. Recent work by Lee and Landgrebe suggests a decision boundary based saliency measure [16].

1.2 Problem Statement

This study will investigate using multiple time series derived from a target time series to improve prediction of a target nonlinear time series. In addition, the usefulness or saliency of various non-target time series in predicting a target time series will be evaluated.

1.3 Summary of Current Knowledge

1.3.1 History. Prior to 1920, forecasting consisted of finding a global average, or possibly a trend in the data and extrapolating the series. Yule developed autoregressive techniques in 1927 in order to improve the prediction of the annual number of sun spots [37]. These methods were based on linear models driven by noise. The equation for an autoregressive moving average (ARMA) model is:

$$x_t = \mathbf{a} \cdot \mathbf{x}_{t-1} + \mathbf{b} \cdot \mathbf{e}_t,$$

where $\mathbf{x}_t = (x_t, x_{t-1}, \ldots, x_{t-(d-1)})$, $\mathbf{a} = (a_1, a_2, \ldots, a_d)$, $\mathbf{b} = (b_1, b_2, \ldots, b_p)$, and $\mathbf{e}_t = (e_t, e_{t-1}, \ldots, e_{t-(p-1)})$. Here d is the order of the autoregressive model, and p is the order of the moving average model. The driving function e_t is usually assumed to be noise.

The linear approach to time series analysis was the prevalent approach until about 1980. However, not all time series could be modeled accurately with linear models. In particular, a low dimensional nonlinear system without noise produces spectra very similar to a noise driven linear model. Put another way, two systems, one linear with noise and the other nonlinear without noise can produce the same spectra. A linear model is completely characterized by the spectra of the modeled data [36:16]. So two very different time series, coincidentally having the same power spectra, would be modeled with the same ARMA model. Figure 1 shows two time series, one linear and the other nonlinear with the same power spectra. The phase scrambled series in the Figure lacks any coherent nonlinear functionality. Linear models are likely to provide the best prediction results for this type of series. The Lorenz time series is the result of low dimensional nonlinearity and nonlinear approaches as discussed here should be able to provide better prediction results than a linear model. Real time series are seldom this easy to classify as linear or nonlinear.

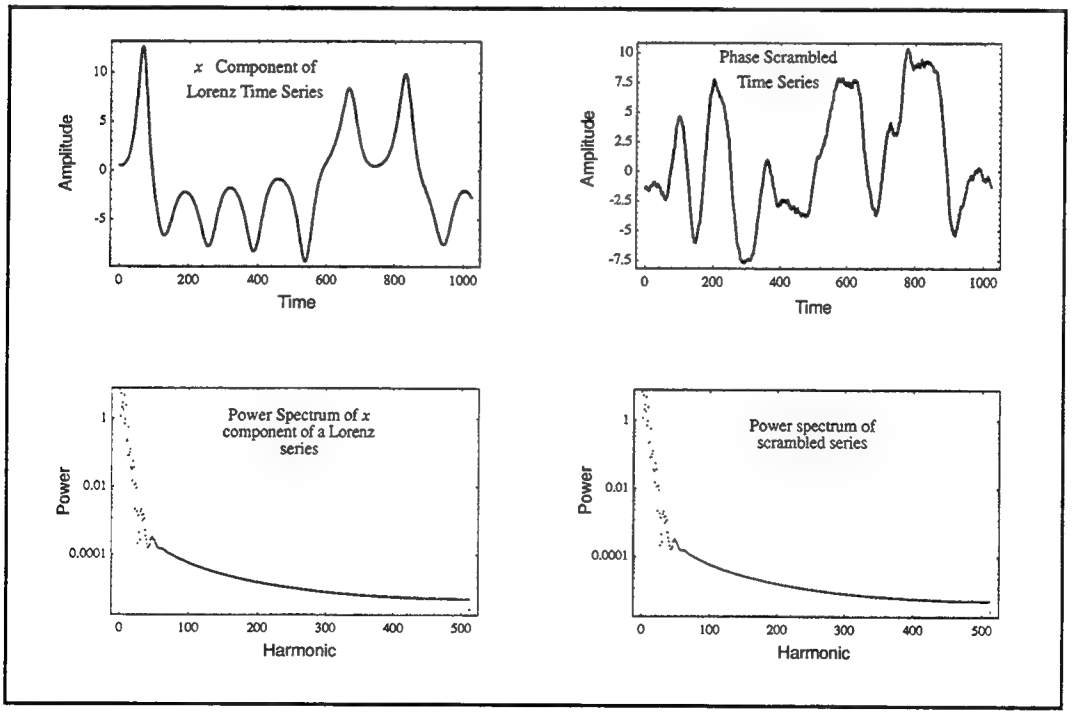

Figure 1. Comparison of the time series and power spectrum of a Lorenz time series and a phase scrambled copy of the same series.

Applying ARMA techniques to nonlinear time series often resulted in time series being classified as random or stochastic, when a low dimensional nonlinear deterministic model could more accurately predict the series.

Around 1980, the increased availability of powerful computers allowed researchers to examine the time series that defied prediction with linear models with methods more appropriate for nonlinear processes. State space reconstruction via time delay embedding was one new twist in time series prediction. This approach uses ideas from differential topology and dynamical systems theory to recognize a time series that is the result of deterministic nonlinear governing equations [36:2]. If a time series is the result of some set of nonlinear deterministic governing equations, state space reconstruction provides a better understanding of the underlying dynamics.

The other new approach that appeared around 1980 was machine learning. Neural networks are an example of this approach. An artificial neural network's ability to "adaptively explore a large space of potential models" is particularly appropriate for modeling of nonlinear time series [36:3].

1.3.2 Sante Fe Time Series Competition. In order to stimulate interest and communication between nonlinear time series researchers, the Sante Fe Institute sponsored a time series competition in 1990. The Institute compiled seven nonlinear time series and requested that participants attempt to predict a short extension of the time series. The participants' results were compared to the actual continuations of the time series. Two participants, Tim Sauer and Eric Wan turned in the best prediction results in the competition [36:8]. Sauer used an enhanced state space approach to accurately predict the short term behavior of a low dimensional chaotic time series. Wan used neural nets to accurately predict the short term behavior of the same time series.

1.4 Scope

Multi-layer perceptron networks will be used to predict the daily closing S&P 500 futures price. Feature saliency techniques will determine which indicator time series are most useful in predicting the target financial series. Candidate indicators include RSI, 4, 9 and 18 day moving averages and 14 and 25 day stochastics. Ruck's saliency metric and a new decision boundary based metric will be compared for determination of the most useful set of financial indicator data. A partial enumeration by indicator feature subset will also be used to determine the usefulness of the indicators for accurate price prediction.

1.5 Thesis Organization

The following chapter details the theory behind state space reconstruction and neural network methods of time series prediction. Although the bulk of this work is concerned with neural-networks, state space reconstruction provides some justification for the use of historical time series data for prediction. Details of several saliency methods are provided here also. Chapter III describes the implementation of these techniques. The results of the application of these techniques to the Standard and Poor's 500 futures data are presented and discussed in Chapter IV. Chapter V summarizes the results of this investigation and suggests areas for further research.

1.6 Summary

Financial time series are representative of nonlinear time series. This thesis investigates multivariate extensions to neural network techniques to accurately predict time series of this type. The results of this investigation should be applicable to other time series that are the result of nonlinear dynamics.

II. Theory

2.1 Introduction

Chapter I was a short overview of nonlinear time series techniques. The goal of this chapter is first to continue to motivate the nonlinear time series problem. After a brief introduction, this chapter will delve into the underlying theory behind two of the methods presented in Chapter I, state space reconstruction and neural networks. The second half of this chapter presents background on feature selection techniques.

A nonlinear time series is characteristic of some nonlinear process. One can classify a time series as nonlinear versus linear by assessing the adequacy of a linear model for prediction of the series. If linear techniques fail to accurately model a time series, the series is classified as nonlinear. Nonlinear time series are typified by financial time series, such as stock market prices.

A second example of a nonlinear time series is the position of an aircraft. Complete position specification requires at least three parameters, aircraft altitude, aircraft latitude and aircraft longitude. These three parameters sampled in time are a multivariate time series. The sponsor for this research, Wright Labs Armaments Division, is keenly interested in improving modeling and prediction of aircraft position time series. Improved prediction of time series results in more accurate tracking of friendly aircraft and more accurate targeting of non-friendly aircraft.

2.2 Scope of Review

The history of linear time series analysis reveals that multivariate time series analysis techniques are built on a firm univariate foundation. Stated another way, multivariate time series analysis methods are extensions of univariate methods [4:401]. The next section summarizes the univariate foundation for nonlinear time series analysis.

The next two sections highlight important aspects of the theory state space reconstruction and neural network techniques as applied to time series prediction.

2.3 *State Space Reconstruction*

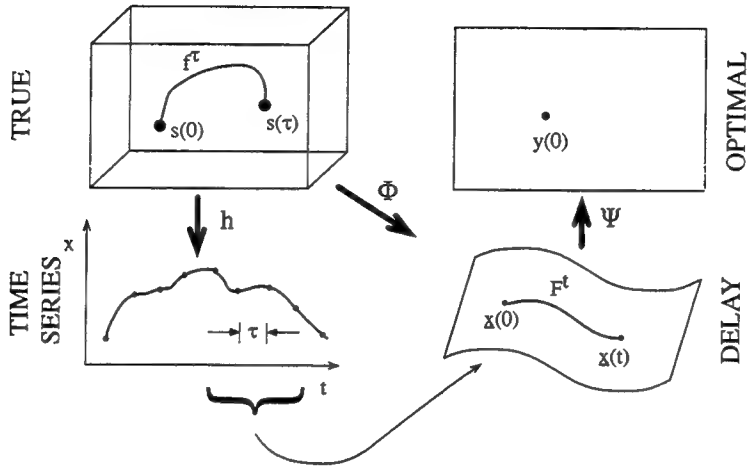

Figure 2. Schematic depiction of the reconstruction problem.

2.3.1 Basic Premise. State space reconstruction methods assume that some number of past samples of a time series, *m*, is sufficient to completely characterize the current state of the underlying process dynamics. Referring to Figure 2, $s(0)$ and $s(\tau)$ are the true state of the dynamical system f^τ at times 0 and τ. The notation $f^t(s(0))$ refers to the state at time t later than the state $s(0)$. A measurement function h relates the time series to the dynamical system by

$$x(t) = h(s(t)).$$

State space reconstruction attempts to reconstruct the true dynamics given only $x(t)$, the time series. Assuming that the past and future of a time series hold information about the state of the unobserved dynamical system, this information can be represented by a *delay vector*

$$\underline{x}(t) = (x(t + \tau m_f), \ldots, x(t), \ldots, x(t - \tau m_p)) .$$

The m-dimensional delay vector includes m_f future samples, and m_p past samples, as well as the current sample $x(t)$. For time series prediction, $m_f = 0$. The separation between samples τ is the lag time.

The mapping Φ of the states of a d-dimensional dynamical system into m-dimensional delay vectors

$$\Phi(s) = \left(h(f^{\tau m_f}(s)), \ldots, h(s), \ldots, h(f^{-\tau m_p}(s))\right)$$

is a delay reconstruction map (Refer to Figure 2). Here $m = m_p + m_f + 1$. The notation $f^{-\tau m_p}$ implies that the underlying mapping f^{τ} is inverted and applied m_p times, resulting in the state that occurred m_p iterations *before* the current state, s. Similarly, $f^{\tau m_f}$ implies that the underlying mapping f^{τ} is iterated m_f times resulting in the state m_f iterations after the current state s.

A theorem by Takens provides that Φ is an embedding as long as $m > 2d + 1$ [30]. An embedding is a smooth one-to-one mapping with a smooth inverse. The space of reconstructed vectors will have smooth dynamics F:

$$F^t(\underline{\mathrm{x}}) = \Phi \circ f^t \circ \Phi^{-1}(\underline{\mathrm{x}})$$

if Φ is an embedding. Here, $\circ$ is a composition operator. With reference to Figure 2, the delay embedding function F^t which maps a point from $\underline{\mathrm{x}}(0)$ to $\underline{\mathrm{x}}(t)$ is a composition of mappings. First, the inverse of the mapping from the true dynamical state to the delay embedding state (Φ) operates on $\underline{\mathrm{x}}(0)$. The underlying mapping f^t operates on this first result, and results in a new dynamical state $s(\tau)$. The delay reconstruction mapping, Φ operates on $s(\tau)$ mapping back to delay reconstruction space, $\underline{\mathrm{x}}(t)$.

These results provide the basis for state space reconstruction. Given m past samples of a times series, where m is the embedding dimension of this time series, we can completely specify the position of the underlying dynamic system in reconstruction space. This position in reconstruction space can be exploited for prediction.

The final block in Figure 2 labeled *OPTIMAL* represents some optimal coordinate transformation, Ψ, from the delay embedding space. Examples of such transformations include principal value decomposition and differentiation. The notion of an optimal transformation is highly dependent on the application. An example of a transformation of this type is Sauer's use of principal components analysis for the prediction task discussed in the next section.

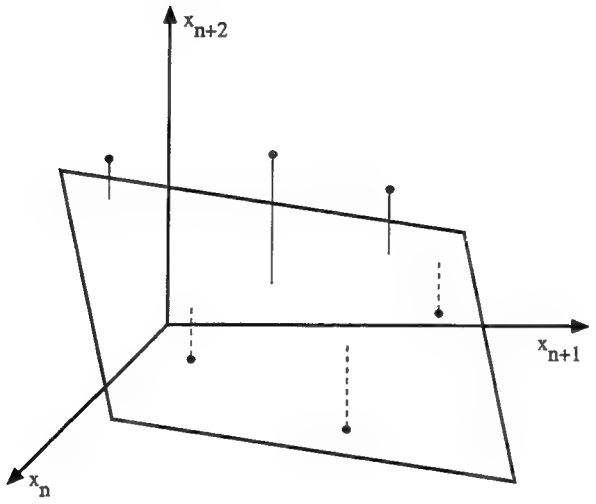

Figure 3. Fitting a hyperplane to the nearest neighbors.

2.3.2 Local Linear Prediction. One method of exploitation of the reconstructed state space is local linear prediction. This technique requires an estimate of m, the embedding dimension. Methods of estimating m are documented in [12, 15, 32]. Given some candidate m-tuple, a search of the past history of the time series for similar m-tuples is conducted. The similar m-tuples are the nearest neighbors to the candidate m-tuple. Regression techniques are then used to fit a hyperplane to these nearest neighbors (Refer to Figure 3 for $m = 3$). The candidate m-tuple is projected onto this hyperplane, resulting in a prediction for the observation following the candidate m-tuple. The method as presented here was published by Farmer in [8:846]. Casdagli [5:308] expanded upon the local linear idea with his Deterministic versus Stochastic (DVS) algorithm.

The local linear prediction method has proven highly successful in predicting low dimensional nonlinear time series. Sauer was able to successfully predict Lorenz-like chaos

in lasers using local linear prediction with several embellishments [26] for the Sante Fe Time Series Competition. He optimized the basic local linear prediction ideas to increase prediction accuracy.

In order to increase prediction accuracy, Sauer interpolated the time series. A Fourier polynomial of degree n,

$$p_n(t) = \alpha_0 + \sum_{j=1}^{n} \left(\alpha_j \left[cos(t) \right]^j + \beta_j \left[sin(t) \right]^j \right),$$

was fit to a section of the time series, and the fit polynomial was sampled at equally spaced intervals. Details on this type of interpolation can be found in [1:177]. With reference to Figure 2 this transforms h before the delay embedding is formed. Figure 4 depicts the effect of this upsampling. The lines with arrowheads represent trajectories in reconstruction space. In addition, he retained only one nearest neighbor from each nearest trajectory [26:181].

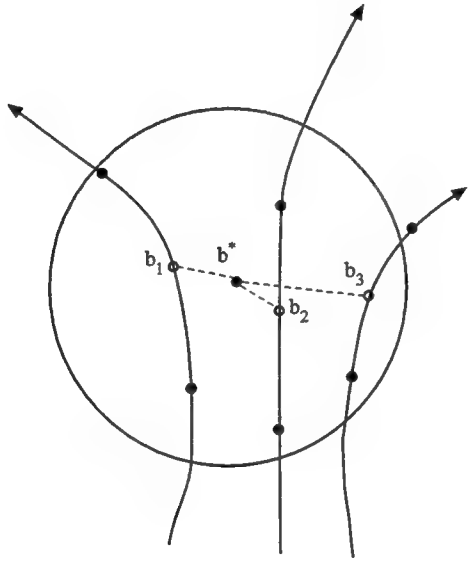

Figure 4. The candidate point $\mathbf{b}^*$ along with its nearest neighbors (solid dots) in reconstructed state space. The open dots are neighbors created by interpolating the time series. Using the interpolated series allows points nearer to the candidate point to be found. Only one neighbor from each trajectory is chosen.

Sauer used singular value decomposition to determine the nearest neighbor's direction of maximum variance. He created a linear model by projecting the nearest neighbors onto this

direction of maximum variance. The candidate m-tuple is also projected onto this direction of maximum variance, and the linear model is evaluated at the point of projection. Sauer assumes the other less variant components (smaller eigenvalues) add only noise to the problem [26:184].

2.3.3 Recent Research. This section will briefly discuss two current research areas in state space reconstruction for prediction of time series.

Two areas of recent research in state space reconstruction methods are more accurate determination of the minimum embedding dimension, m, and surrogate data methods. Typically, a dimension estimation algorithm such as [12] is used to estimate m. These dimension estimation algorithms generally use the some measure of the number of nearest neighbors of a given dimension within a hypercube to calculate a fractal dimension. These methods are not foolproof. Theiler provides a good introduction to these dimension estimation methods [32].

Matt Kennel suggests a significant improvement to this algorithm, based the elimination of false nearest neighbors [15, 14]. A false nearest neighbor is defined as a point that is near in m dimensional Euclidean space, but not near in $m+1$ dimensional Euclidean space. The key to this method is that true nearest neighbors remain close when the dimension is increased.

Theiler and others have taken a bootstrap approach to classifying time series as linear or nonlinear [33]. The approach here is to generate random data with similar characteristics to the target time series. For instance, a random series with the same power spectrum as a target series can be generated. Figure 1 is an example of scrambling one component of a Lorenz series. Then a statistic such as embedding dimension is calculated on both the target series and numerous random or *surrogate* series. A statistical procedure is applied to determine if the target series' statistic is significantly different than the surrogate data statistic. If the surrogate and target statistics are not significantly different, the conclusion is that the target series is linear with noise, as opposed to nonlinear. These surrogate techniques are useful for determining the aptness of nonlinear techniques to a particular series.

2.4 *Neural Networks*

2.4.1 Basic Theory. Artificial neural networks were devised to mimic the response of a true biological neuron. The most popular model of a neuron is a perceptron, schematically depicted in Figure 5. A perceptron consists of several input sensory nodes fully connected to an associative node. The synapses or connections are called weights, and initially set at random values uniformly distributed from -1 to 1. The associative nodes produce an output when sufficient stimuli arrive. These stimuli are sensed by the input nodes, multiplied by the appropriate synapse weights, and summed at the associative nodes. The associative node acts on this weighted sum usually with some nonlinear function $f()$, often a sigmoid, and produces an output. The perceptron is able to learn by adjusting the weights so a given stimulus produces the desired response [25, 23].

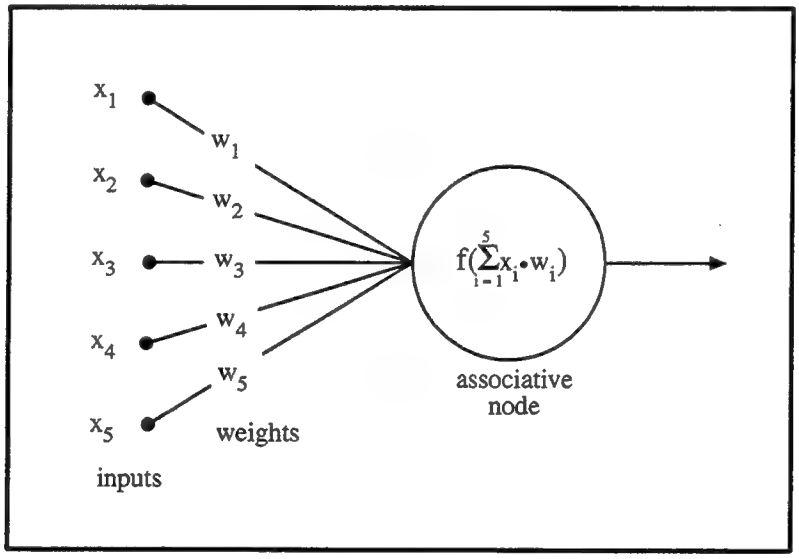

Figure 5. Schematic depiction of a perceptron.

A multilayer perceptron (MLP) is composed of layers of perceptrons. All input nodes are connected to all layer one nodes. All layer n nodes are connected to all layer $n+1$ nodes. Like a perceptron, an MLP can be trained to produce a desired output. In fact, an MLP is capable of approximating any continuous nonlinear function, given enough nodes, enough layers and enough training data [23:87]. This ability to model any nonlinear function is not without an associated cost. Training an MLP is more involved than training a simple perceptron. One

method of setting the weights or training the network is called backpropagation of error. In backpropagation, the difference between the desired output of the MLP and the observed output is propagated back through the network, and the weights are adjusted to minimize this difference [21].

A neural network's function modeling capability allows neural networks to approximate the underlying nonlinear function of a nonlinear time series. Stright used a generic MLP with m inputs of delayed time series values to predict incommensurate sine wave data and Glass-Mackey data with some success in 1988 [28:5-1].

2.4.2 Time Delay Neural Networks. Waibel modified an MLP to use only temporally related information to predict Japanese phonemes in 1989 [34]. He constrained weights on his network to remove any non-temporal information from the network. Waibel called his net a Time Delay Neural Network (TDNN).

Wan improved on Waibel's work with his linear filter approach to the temporal constraining [35]. His network is equivalent to Waibel's TDNN, but Wan's Finite Input Response (FIR) method is both conceptually simpler and numerically more efficient. The FIR approach replaces nodes with FIR Filters, and uses a temporal backpropagation algorithm to train the network.

Using his FIR network, Wan produced the best prediction results on the laser data of the Sante Fe competition. Wan's prediction root mean square error (rmse) was one fourth of Sauer's state space reconstruction prediction rmse [36:10]. Although Wan's results were better than Sauer's, both methods produced short range predictions that essentially duplicated the data [36:9].

2.4.3 Recent Research. Recent research in time series prediction with neural networks has taken a different approach to network architecture than that used by Wan and Waibel. Instead of connecting the input layer only to the hidden layer nodes and the hidden layer, Holt and others recommend making direct connections from the input layer to the

output layer, and summing the products of these directly connected inputs and their weights with the nonlinear output [13, 18]. This architecture, called Direct Linear Feedthrough (DLF), enhances the modeling of the linear portion of the mapping from input to output. This approach appears, in principle, to be very similar to using a neural network to predict the residuals from an autoregressive model.

2.5 Feature Selection

Feature selection is the process by which a large set of candidate features is reduced to a smaller set. In neural network terms, a feature is an individual data element fed into an input node. Typical features for S&P 500 prediction include historic data, and indicators. The candidate features for estimation of tomorrow's closing price of the S&P 500 futures market might include:

1. Today's closing price.
2. Yesterday's closing price.
3. The closing price two days ago.
4. The closing price three days ago.
5. The closing price four days ago.
6. The closing price five days ago.
7. Today's 10 day moving average of the closing price.
8. Yesterday's 10 day moving average of the closing price.
9. Today's RSI.

The estimate for tomorrow's closing price may be strongly related to several of these features, while others are not related. Feature selection techniques are aimed at partitioning the feature set into the important or *salient* features and the unimportant features.

Feature selection has several benefits. First, it can reduce the amount of data that must be recorded. Second, more features usually lead to more parameters that must be estimated. In order to have a valid general model, the number of training or sample data points required is directly related to the number of input features [9]. In other words, fewer features means

fewer samples are required to build a model. Finally, if a feature is unimportant for some classification or regression task, the network is essentially being fed noise in addition to the signal of the important features. Feature selection helps increase the signal to noise ratio that the network sees.

There are several approaches to neural network feature selection. Steppe does an excellent job of surveying these techniques [27]. Feature selection techniques falls into one of three classes. The first class of techniques involve a search for relevant feature subsets. The second class of techniques, saliency metrics, is concerned with ranking the individual features. The third class of techniques is concerned with screening irrelevant features. These screening techniques are used to determine the partition in the ranked features between the salient and non-salient features. One method of screening irrelevant features is including a noise feature in the feature set. Feature saliency metrics can then be calculated and all features less salient than the noise feature are assumed to be irrelevant.

2.5.1 Steppe Approach. Searching for relevant feature subsets can be computationally intractable for feature sets of any size. A complete examination of every possible subset requires examining $2^{features}$ subsets. One method that avoids the intractability associated with examining every feature subset is sequential selection. These techniques trade optimality for tractability. Stated another way, sequential selection techniques are not guaranteed to find the optimum feature subset. Steppe implemented a backward sequential selection algorithm for neural networks [27:148]:

1. Set $k = M$, where M is the total number of candidate features.
2. Choose a significance level α for feature elimination.
3. Estimate the full model with k candidate features. Associated with this full model is SSE_F for univariate models and T_F for multivariate response models.
4. Set $p = k - 1$, where p is the number of features in the reduced model.

5. Estimate all models with p features which do not include any previously eliminated features. Associated with each of the k reduced models is SSE_R for univariate models and T_R for multivariate response models.
6. Compute a likelihood ratio test statistic for each of the k models of p features.
7. Select, as a candidate feature for elimination, the feature which when removed produces the model with the lowest likelihood ratio test statistic L.
8. If $L < threshold$, eliminate the candidate feature, set $k = k - 1$ and go to Step 3. Otherwise go to Step 9 and do not eliminate the candidate feature.
9. Stop.

2.5.2 Saliency Metrics. The second class of feature selection techniques requires the computation of saliency metrics. Several saliency metrics have been proposed by Ruck, Priddy and Tarr [24, 22, 31]. Steppe demonstrated the equivalence of these metrics [27]. This section will briefly explain the Ruck saliency metric for MLP networks. Details of derivation and statistical background may be found in [27, 24].

The Ruck saliency metric sums the partial derivatives of the neural network output with respect to a given input. Assuming, for instance, a two class problem with four inputs, the saliency metric for input x_1 (in theory) integrates the partial derivative of each network output with respect to x_1 over the input space. In reality, the integration is approximated with a summation over either each input vector or a random sampling of the input vectors. The result is a measure of the usefulness of each input feature to determination of the correct output class.

The Ruck saliency metric requires the summation of the partial derivatives of a networks outputs with respect to a given input over some approximation of the input space. The Lee and Landgrebe method requires calculation of the partial derivatives only at decision boundaries. A decision boundary is the locus of points in input space, separating one class from another. For a neural network, the decision boundary is the locus of points where the network's outputs are equal. A brief explanation of the Lee and Landgrebe approach follows.

2.5.3 Lee and Landgrebe Approach. The Karhunen-Loeve transformation (KLT), also called principal value decomposition, is one technique that can be used for feature extraction. The KLT method extracts the eigenvalues and eigenvectors from the full data covariance matrix. KLT is optimum for data reconstruction, or stated another way, no linear transformation of the original data is superior to the dominant KLT vectors minimizing reconstruction error of the data in a mean square sense. While useful for reducing data transmission requirements, KLT is not optimal in terms of class discrimination [17:388].

Lee and Landgrebe suggest a feature extraction technique based on decision boundaries [17]. Their method is based on the idea that the important information in a classification problem is perpendicular to the classification boundary. Figure 6 depicts a portion of a decision boundary. A feature is *discriminantly informative* if moving in the direction of the feature would have affected the classification of at least one observation [16:434]. If a feature has no effect on the classification of any of the observations, the feature is *discriminantly redundant.* To affect classification, a feature must have a component perpendicular to a decision boundary.

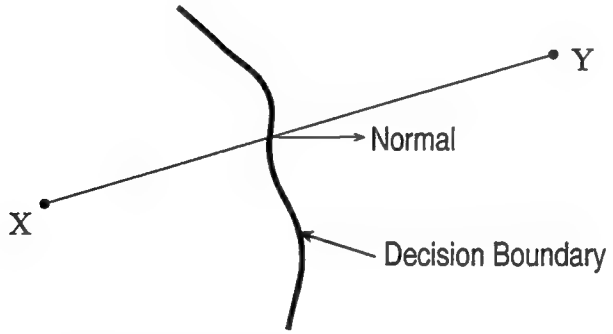

Figure 6. A portion of a decision boundary between point X of class 1 and point Y of class 2, and the normal direction at the point where the line connecting X and Y crosses the decision boundary.

Instead of manipulating a data covariance matrix, Lee and Landgrebe create an effective decision boundary feature matrix (EDBFM) [17:392]. The EDBFM is the outer product of the normals of all the effective decision boundary points. An effective decision boundary point is the point on a line between two observations that are classified differently, where the discriminant function value is zero or near zero. The normals are determined by taking

the gradient of the discriminant function at the effective decision boundary point. The outer product produces a dispersion (covariance) matrix of the normals.

The Lee and Landgrebe procedure for creating the EDBFM, for the 2 class case [16:436] is:

1. Classify the training data.
2. For each sample correctly classified as class c_1, find the nearest sample correctly classified as class c_2. Similarly, for each sample correctly classified as class c_2, find the nearest sample correctly classified as class c_1.
3. Connect the pairs of samples found in Step 2. The line connecting the pair of samples must pass through the decision boundary. Find the point along the line that is at the decision boundary or within a threshold of the decision boundary.
4. At each point $\mathbf{x}_i$ found in Step 3, estimate the unit normal vector $\mathbf{N}_i$ by $\mathbf{N}_i = \nabla h(\mathbf{x}_i)$, where $h(\mathbf{x})$ is the discriminant function, and the gradient is taken with respect to the inputs and normalized to unit length.
5. Estimate the decision boundary feature matrix using the normal vectors found in Step 4.
6. Select the eigenvectors of the decision boundary feature matrix as new feature vectors according to the magnitude of the corresponding eigenvalues.

The Lee and Landgrebe procedure is similar to KLT in its use of orthogonal decomposition. Whereas KLT provides the best decomposition for reconstruction of the data, the decision boundary technique provides the best decomposition for reconstruction of the decision boundary [17:393]. The eigenvalues of the EDBFM provide a measure the total projection of the normal vectors on the individual eigenvectors. This information, along with the eigenvector directions can be used to rank the input features based on their contribution to the eigenvalues.

2.6 *Summary*

Multivariate nonlinear prediction depends on good univariate nonlinear prediction methods. This review has detailed two successful approaches to univariate nonlinear prediction. The two approaches, state space reconstruction and time delay neural networks are the cornerstones of nonlinear time series prediction. The next chapter describes extensions to the decision boundary based techniques as well as the application of neural net prediction techniques and saliency metrics to the S&P 500 and the indicator data.

III. Methodology

3.1 Introduction

The last chapter presented the background behind nonlinear prediction techniques and outlined several methods of determining feature saliency. This chapter picks up where the saliency material from Chapter II left off. First, the Lee and Landgrebe decision boundary techniques are extended for an artificial neural classifier and for use without a classifier. Then, a decision boundary based saliency metric is derived and compared to previous saliency metrics.

Section 3 presents the rationale for an indicator feature subset enumeration approach to be used in Chapter IV. The last section discusses the preprocessing of the S&P 500 futures price data. First, formulae and justification for candidate indicators are provided. The chapter closes with a discussion of the detrending and normalization applied to the data to make it compatible with a neural network.

3.2 Saliency

3.2.1 Lee and Landgrebe for Neural Nets. Lee and Landgrebe provide a method for decision boundary based feature extraction using a Gaussian classifier in [17], and using Parzen density estimation in [16]. This section extends their methods to neural network based classification.

Feature extraction is not feature *selection*, but it can be argued that in order for a feature to be salient, it must contribute to the dominant extracted features. A saliency metric for the original features *based on the extracted features* will be discussed.

3.2.1.1 Neural Network. The decision boundary feature extraction procedure for neural networks follows the Lee and Landgrebe procedure as discussed in Chapter II. The steps are repeated here for reference.

1. Classify the training data.

2. For each sample correctly classified as class c_1, find the nearest sample correctly classified as class c_2. Similarly, for each sample correctly classified as class c_2,find the nearest sample correctly classified as class c_1.

3. Connect the pairs of samples found in Step 2. The line connecting the pair of samples must pass through the decision boundary. Find the point along the line that is at the decision boundary or within a threshold of the decision boundary.

4. At each point $\mathbf{x}_i$ found in Step 3, estimate the unit normal vector $\mathbf{N}_i$ by $\mathbf{N}_i = \nabla h(\mathbf{x}_i)$, where $h(\mathbf{x})$ is the discriminant function, and the gradient is taken with respect to the inputs and normalized to unit length.

5. Estimate the decision boundary feature matrix using the normal vectors found in Step 4.

6. Select the eigenvectors of the decision boundary feature matrix as new feature vectors according to the magnitude of the corresponding eigenvalues.

This section details determination of the decision boundary points, and calculation of the gradients at these points.

3.2.1.2 Decision Boundary. After training a network and obtaining acceptable results, the weights are extracted from the net and a discriminant function is created. In general, for a neural net with a single hidden layer, the output vector is:

$$\mathbf{y} = \mathbf{f}_o(\mathbf{w}_2 \cdot \{\mathbf{f}_h(\mathbf{w}_1 \cdot \{\mathbf{x}; 1\}); 1\}),$$

where $\mathbf{f}(\mathbf{a})$ applies the node function to each element of $\mathbf{a}$ and results in a vector. The hidden node function, $\mathbf{f}_h$, and the output node function, $\mathbf{f}_o$ need not be the same. The semicolon is a concatenation operator; $\{\mathbf{x}; 1\}$ appends a 1 to the end of the $\mathbf{x}$ vector. $\mathbf{w}_1$ is the matrix of weights connecting the inputs to the hidden nodes. The $\mathbf{w}_1$ matrix has *number of inputs +1* columns and *number of hidden nodes* rows. $\mathbf{w}_2$ is the matrix of weights connecting the hidden

layer to the output nodes. $\mathbf{w}_2$ has *number of hidden nodes +1* columns and *number of output nodes* rows.

The output vector is used to create a discriminant function. For a two class problem with two output nodes, a common discriminant function is

$$h(\mathbf{y}) = -ln\frac{y_1}{y_2},$$

where y_1 is the first output.

The line connecting two points, $\mathbf{x}_1$, and $\mathbf{x}_2$ classified differently, is parameterized

$$\mathbf{x}(t) = \mathbf{a}t + \mathbf{x}_1,$$

where

$$\mathbf{a} = \mathbf{x}_2 - \mathbf{x}_1.$$

At some point on this line, $\mathbf{x}(t)$, the discriminant function is at or near zero. A numeric root finding method such as the secant method can be used to solve for the root, t_0. $\mathbf{x}(t_0)$ is the decision boundary point.

Once the decision boundary points are determined, the next step is to calculate the gradient of the discriminant function at the decision boundary points with respect to the inputs.

3.2.1.3 Gradient. Calculating the gradient of the discriminant function is just an extended application of the chain rule. For a two class problem with *n* inputs, start with:

$$h(\mathbf{y}) = -ln\frac{y_1}{y_2}, \tag{1}$$

$$\mathbf{y} = \mathbf{f}_o(\mathbf{w}_2 \cdot \{\mathbf{f}_h(\mathbf{w}_1 \cdot \{\mathbf{x}; 1\}); 1\}). \tag{2}$$

Assuming the the network has two outputs,

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}.$$

The gradient of the discriminant is

$$\nabla h(\mathbf{y}) = \begin{bmatrix} \frac{\partial h(\mathbf{y})}{\partial x_1} \\ \frac{\partial h(\mathbf{y})}{\partial x_2} \\ \vdots \\ \frac{\partial h(\mathbf{y})}{\partial x_n}. \end{bmatrix} \tag{3}$$

Since all the partials will be calculated in a similar manner, $\frac{\partial h(\mathbf{y})}{\partial x_1}$ will be demonstrated here. In order to determine

$$\frac{\partial h(\mathbf{y})}{\partial x_1} = \frac{1}{y_2}\frac{\partial y_2}{\partial x_1} - \frac{1}{y_1}\frac{\partial y_1}{\partial x_1} \tag{4}$$

we need $\nabla \mathbf{y}$. Taking the gradient of equation 2,

$$\nabla \mathbf{y} = \nabla \left(\mathbf{f}_o(\mathbf{w}_2 \cdot \{\mathbf{f}_h(\mathbf{w}_1 \cdot \{\mathbf{x}; 1\}); 1\}) \right), \tag{5}$$

and using the chain rule,

$$\nabla \mathbf{y} = \mathbf{I}_o \cdot \mathbf{f}_o'(\mathbf{z}) \cdot \mathbf{w}_2 \cdot \mathbf{I}_{h+1} \cdot \{\mathbf{f}_h'(\{\mathbf{x}; 1\}); 0\} \cdot \mathbf{w}_1, \tag{6}$$

where $\mathbf{z}$ is the output from the hidden layer,

$$\mathbf{z} = \{\mathbf{f}_h(\mathbf{w}_1 \cdot \{\mathbf{x}; 1\}); 1\}. \tag{7}$$

The gradient $\nabla \mathbf{y}$ is a *number of outputs* by *number of inputs* matrix. For the two output case, the first column contains the partials with respect to first input. These partials can be

substituted into Equation 4, and the rest of the components of $\nabla h(\mathbf{y})$ can be calculated in the same manner.

This matrix derivation agrees with previous derivations of the individual partials [27:62], and can be easily programmed in a matrix algebra package such as *Mathematica* or *Matlab*. Appendix A includes a *Mathematica* example.

The only remaining step of the Lee and Landgrebe procedure is to calculate the effective decision boundary feature matrix (EDBFM). The gradients at each decision boundary point are normalized to unit length. The outer product of the matrix of these normals (*number of decision boundary points* by *number of features*) with itself results in a positive definite matrix (*number of features* by *number of features*). The eigenvalue analysis of this matrix described in Chapter II results in a ranked list of discriminantly informative directions in feature space.

3.2.1.4 Saliency. Metrics for determining the saliency of features using neural nets generally involve integration of the partial derivatives of the discriminant function over the data space. Since an input feature is only salient if it has components parallel to the significant eigenvectors of the EDBFM, summing the loadings of each input vector over the eigenvectors and weighting by the eigenvalues of that vector should provide an input feature saliency metric that agrees with the other methods.

3.2.1.5 Example. A simple two dimensional two class example will illustrate the decision boundary procedure as applied to neural networks. Two sets of normally distributed data are generated with parameters:

$$M_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & .5 \\ .5 & 1 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & .5 \\ .5 & 1 \end{bmatrix}.$$

Here M_1 is the centroid or mean location of the class 1 data and Σ_1 is the covariance matrix of the class 1 data. The data are depicted in Figure 7. The optimal decision boundary between two Gaussian distributions with the same covariance matrix can be determined using Fisher's discriminant function [6:364]:

$$y = \hat{\mathbf{b}} \cdot \mathbf{x}, \tag{8}$$

where, the class is determined by the sign of y, and

$$\hat{\mathbf{b}} = \mathbf{S}^{-1} \left(\bar{\mathbf{x}_1} - \bar{\mathbf{x}_2} \right), \tag{9}$$

and $\mathbf{x}$ is the candidate input vector. The symbol $\mathbf{S}$ usually represents a pooled covariance matrix, but in the two dimensional example, $\mathbf{S} = \Sigma_1 = \Sigma_2$. The means, $\bar{\mathbf{x}}_1$ and $\bar{\mathbf{x}}_2$ are the class distribution means, M_1 and M_2 respectively. Substituting the parameters, and solving for $\hat{\mathbf{b}}$,

$$\hat{\mathbf{b}} = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}^{-1} \left(\begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 2.0 \\ 1.0 \end{bmatrix} \right) = \begin{bmatrix} -2.0 \\ 2.0 \end{bmatrix}. \tag{10}$$

Substituting back into Equation 8 we get

$$y = -2x_1 + 2x_2. \tag{11}$$

Recalling that the decision boundary occurs where $y = 0$, the equation for the decision boundary is

$$x_1 = x_2. \tag{12}$$

These calculations assume that the actual data have sample covariances and means equivalent to the generating parameters. The sample statistics are approximately the same as the generating parameters.

A neural neural network with 2 inputs, 4 hidden layer nodes and 2 outputs is trained on the data. Figure 8 displays the network architecture. The weights are extracted from this

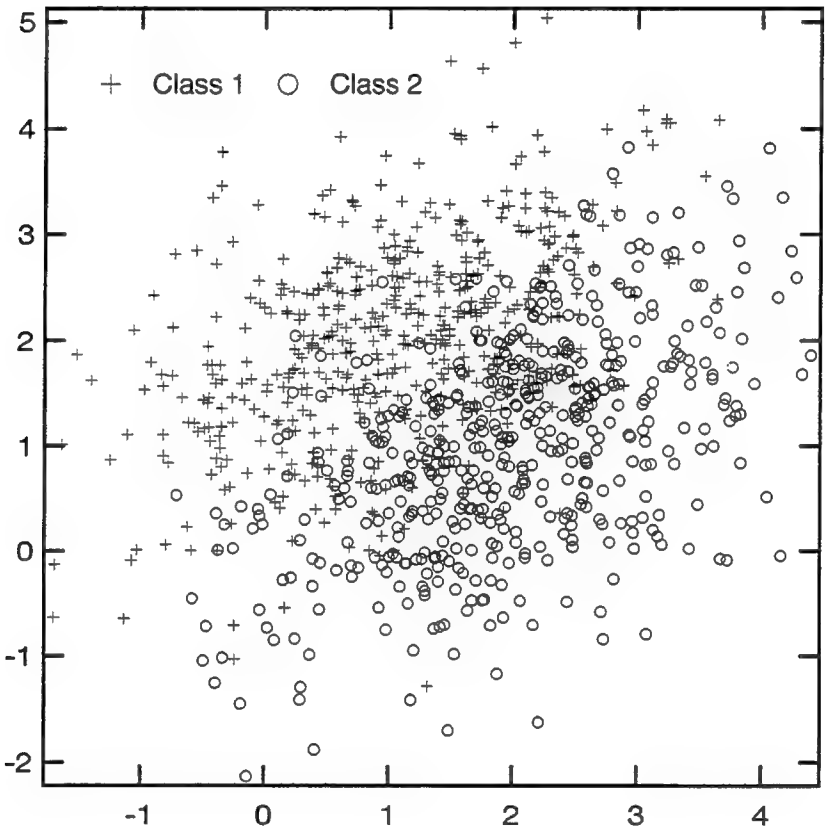

Figure 7. Experimental data distribution.

network and the decision boundary methodology is applied. Details on the calculations using *Mathematica* can be found in Appendix A.

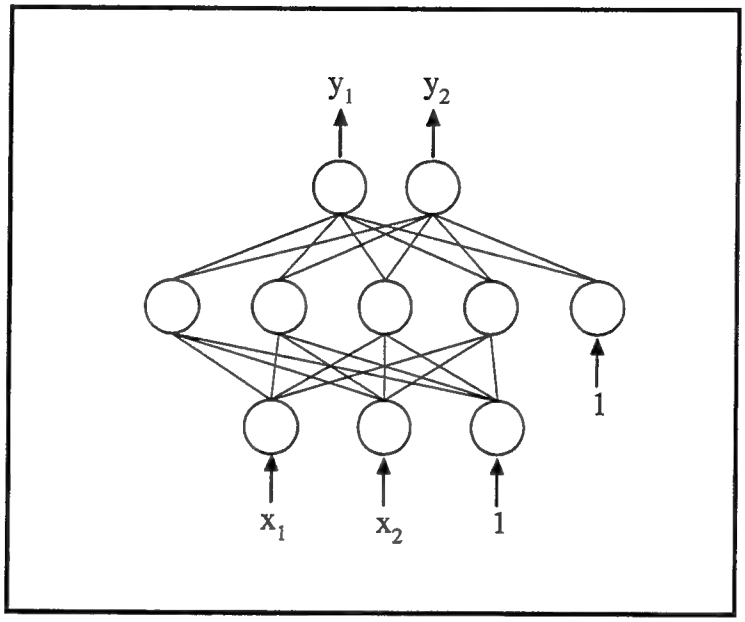

Figure 8. Network architecture for Gaussian distribution discrimination.

Figure 9 shows the neural network decision boundary. The network closely approximates the optimum decision boundary.

The eigenvalues (λ) and the eigenvectors (ϕ) extracted from the EDBFM are:

$$\lambda_1 = 0.9996, \qquad \lambda_2 = 0.0004,$$
$$\phi_1 = \begin{bmatrix} 0.668 \\ -0.744 \end{bmatrix}, \quad \phi_2 = \begin{bmatrix} -0.744 \\ -0.668 \end{bmatrix}.$$

ϕ_1 is the only discriminantly informative direction since $\lambda_1 >> \lambda_2$. These values agree with Lee and Landgrebe's results for a Gaussian classifier [17:395].

The saliency of each input can be calculated by summing the weighted squares of the appropriate eigenvector component. The eigenvalues are the weights. With each input of the

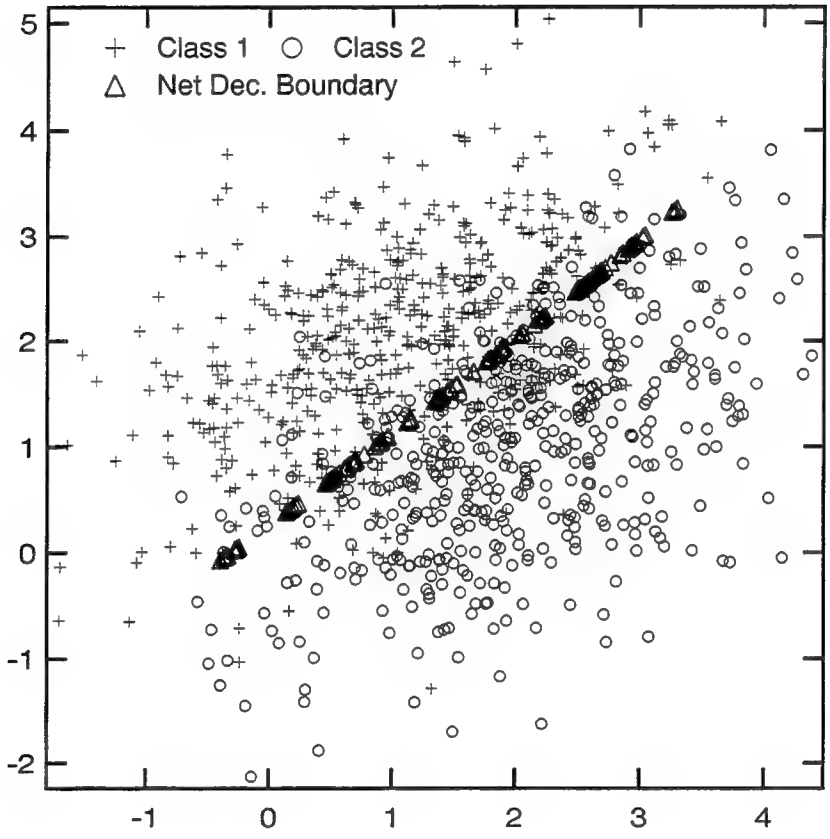

Figure 9. Experimental data distribution with decision boundary.

form:

$$\mathbf{x}_i = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

the saliency metric for the input i, γ_i, is

$$\gamma_i = \sum_{j=1}^{n} \phi_{i,j}^2 \lambda_j. \tag{13}$$

$\phi_{i,j}$ is element i of eigenvector j. The summation is over the number of eigenpairs. Eigenpair with small eigenvalues will contribute very little and can be ignored. Returning to the example, the calculated saliency metrics are

$$\begin{aligned} \gamma_1 &= 0.4463, \\ \gamma_2 &= 0.5535. \end{aligned}$$

The inputs are approximately equally salient. Both inputs contribute to the discriminant function, so neither can be deleted without significantly affecting classification accuracy.

The next subsection is an attempt to apply the basic concepts of decision boundary analysis without using a classifier of any type.

3.2.2 Lee and Landgrebe without a classifier. Lee and Landgrebe's feature extraction procedure depends on the determination of a decision boundary location between two points from different classes. A simple extension of this idea is to assume that some form of decision boundary will correctly separate the labeled data into correct classes. Previously, the decision boundary was determined using a classifier. Here a perfect decision boundary is *assumed.*

Instead of explicitly calculating the gradient at some decision boundary point, the line that connects two close points from different classes is used as an approximation to the decision boundary. As in the previous case, these pseudo-normal vectors are normalized to unit length. The two close points from different classes are chosen in the same manner as in the previous

example, except that all are assumed correctly classified. The EDBFM is created based on the directions between these points, and the eigenvalues are calculated.

This method was initially applied to the same Gaussian data as in the previous section, with parameters:

$$M_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & .5 \\ .5 & 1 \end{bmatrix}$$
$$M_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & .5 \\ .5 & 1 \end{bmatrix}.$$

The resulting eigenpairs are:

$$\begin{aligned} \lambda_1 &= 0.555, & \lambda_2 &= 0.445, \\ \phi_1 &= \begin{bmatrix} 0.619 \\ -0.785 \end{bmatrix}, & \phi_2 &= \begin{bmatrix} -0.785 \\ -0.619 \end{bmatrix}. \end{aligned}$$

These results are not very promising since the optimum classification boundary is one dimensional, and these results suggest a two dimensional decision boundary. The eigenvectors are however, approximately correct.

Referring back to Figure 9, it is apparent that a large number of class 1 data are in the midst of the class 2 data, and vice-versa. The outer product of the normals creates a dispersion matrix of the pseudo-normals, with each pseudo-normal being weighted the same. A means of weighting the more clearly separated points might provide a better characterization of the implicit decision boundary. Any reference here to separated points is intended to mean points in separate classes. Building the EDBFM from the un-normalized pseudo-normal vectors is one means of weighting the more clearly separated points. This method should decrease the effect of the overlapping areas of the distributions. Unfortunately, using the un-normalized vectors will have no impact on the associated nearest neighbor problem. In the simple two Gaussian example, the points in the overlapping area, especially the worst outliers in the

overlap area will have a large number of nearest neighbors, since they are the nearest neighbor to many and perhaps most of the opposite class. Many of the pseudo-normals associated with these extreme outliers will be pointing in a non-discriminative direction. A method for removing these outliers (without using a classification mechanism) is unclear at this point.

The results after removing the normalization are:

$$\begin{aligned} \lambda_1 &= 0.701, & \lambda_2 &= 0.299, \\ \phi_1 &= \begin{bmatrix} 0.726 \\ -0.687 \end{bmatrix}, & \phi_2 &= \begin{bmatrix} -0.687 \\ -0.726 \end{bmatrix}. \end{aligned}$$

Some improvement is seen here. Although λ_1 does not dominate λ_2, it is significantly larger. The eigenvectors are still reasonable.

Intuitively, if the Gaussians are moved further apart, there would be fewer fringe class 1 points mixed in with class 2 and vice versa. A data set was created with the same covariances but with means:

$$M_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, M_2 = \begin{bmatrix} 3 \\ 0 \end{bmatrix}.$$

The results after normalizing the pseudo-normal lengths are:

$$\begin{aligned} \lambda_1 &= 0.731, & \lambda_2 &= 0.269, \\ \phi_1 &= \begin{bmatrix} 0.726 \\ -0.687 \end{bmatrix}, & \phi_2 &= \begin{bmatrix} -0.687 \\ -0.726 \end{bmatrix}. \end{aligned}$$

Since there are fewer fringe points, the normalization of the pseudo-normals does not totally degrade the results. Performing the same experiment without normalization :

$$\begin{aligned} \lambda_1 &= 0.830, & \lambda_2 &= 0.170, \\ \phi_1 &= \begin{bmatrix} 0.748 \\ -0.663 \end{bmatrix}, & \phi_2 &= \begin{bmatrix} -0.663 \\ -0.748 \end{bmatrix}. \end{aligned}$$

The non-normalized pseudo-normal results are again superior.

Although the non-classifier results are not as clearly one-dimensional as the classifier results, the non-classifier approach can provide a classifier independent initial cut at the important features that can be extracted from the data. With the same saliency metric as defined previously, these results can also provide initial saliency results. Some of the ability to discriminate between the important eigenvectors, and indirectly, the important features, is lost, but the possible savings provided by extracting features before classification counter this loss. Further research is required to find a mechanism to reduce the impact of outliers in the midst of the other class. Possible mechanisms might involve some use of the distribution of pseudo-normal lengths or use of the relative numbers of nearest neighbors to each point of a class.

3.2.3 Lee and Landgrebe for Regression. In order to use the Lee and Landgrebe technique for feature extraction in the time series prediction task, the technique must be modified for regression. Regression, also called function mapping, is the task of predicting a value as opposed to predicting a class in classification. Unfortunately the notion of a decision boundary in the regression case is unclear. Instead of a boundary between two classes, regression produces a contour map of values over the input space. Some level contour on this contour map must be chosen as a decision boundary of sorts. The value of the contour chosen here is zero. This choice is motivated by the importance of trend in the stock market. If the target data are price first differences, the null or zero contour line defines the boundary between the up and down trend of the market. Even if the data are not first differences, with any detrending method the zero contour marks the difference between reducing the prediction error and increasing the prediction error. If the target data are not detrended, the choice of zero is not appropriate, since a zero output may not even occur.

Indicator	Number of associated features
RSI	1
4-day Moving Average	2
9-day Moving Average	2
18-day Moving Average	2
10-day Stochastics	4
25-day Stochastics	4

Table 1. Indicator feature subsets.

3.3 Indicator Feature Subset Enumeration

In many applications subsets of the input features are assumed to be relevant. Steppe's algorithm selects individual features for removal from the candidate data set [27]. Often domain knowledge leads one to assume that a certain subset of features may be relevant to the problem. In the financial markets, technical analysts rely on indicators to predict future prices. Technical market analysts often use more than the daily value of an indicator; they may take the indicators' trend into account as well. In fact, the two previous values of an indicator may be relevant to the price prediction. These two indicator value features represent a logical subset of the input features. If an indicator is not useful to a neural network the entire associated subset can be dropped.

The advantage of using predetermined subsets of features is that it can reduce the intractability associated with analysis of all subsets, since the number of subsets that must be evaluated for complete enumeration is 2^k where k is the number of discrete entities, either features or, in this case, feature subsets. For the S&P 500 task there are 21 individual features. Some of these features are members of indicator subsets. We can reduce k from 21 to 6 by enumerating based on indicator subsets. This results in a reduction from 2,097,152 subsets to 64 subsets.

The indicator feature subsets and the associated number of elements are shown in Table 1. These indicators are defined in the next section.

3.4 Data Preprocessing

This section addresses the preprocessing applied to the S&P 500 futures data prior to prediction. An explanation of candidate technical market indicators is followed by a discussion of the specific detrending and normalization techniques employed in this research.

3.4.1 Technical Indicators. Many stock market indicators exist. The popular literature in this area [2, 7] seems to suggests a strong reliance on trend indicators such as moving averages and overbought/oversold indicators such as Wilder's RSI and Stochastics. Overbought/oversold indicators try to characterize the momentum of the the market. The basic idea is that an oversold market should soon experience a price rise, and an overbought market is expected to experience a decrease in price. Each of these indicators is dependent on some time parameter. The goal here is to investigate prediction of the S&P 500 daily closing price. Discussions with traders resulted in the selection of the 14 day RSI, 4, 9 and 18 day moving averages and 10 and 25 day stochastics as candidate indicators.

The data set provided includes daily pricing information for the S&P 500 futures. Each line of the data set contained data from one trading day. For each trading day the data includes seven items. The first field of the daily data is a date stamp, indicating the trading day. The second field contains the opening price for the day.The third field is the highest price the commodity attained during the trading day. The fourth field holds the commodity's lowest daily price. The fifth field is the closing price of the day. The last two fields contain volume and open interest information. The next section explains how each candidate indicator is calculated from the data set.

3.4.1.1 Moving Averages. A moving average is simply an average of a fixed number of past samples. Assuming we have ten days of data, the ten day moving average of the closing price for the tenth day is simply the sum of the ten days closing price divided by ten. Figure 10 shows the moving average results for the candidate data set. One trading technique generates buy or sell signals based on the crossing of the quicker moving average (4 day) over the longer moving average (9 day). The general idea here is to buy into the commodity when

the short trend crosses the long trend from below, and sell when the shorter average crosses the long trend from above. Other techniques use the direction a trend is moving to generate buy and sell signals. In order to calculate a trend, two data points are required. Based on this information, in order to predict Wednesday's price, a minimum of Monday's and Tuesday's moving average information is required. Each moving average feature we evaluate contains information for the two days prior to the prediction day.

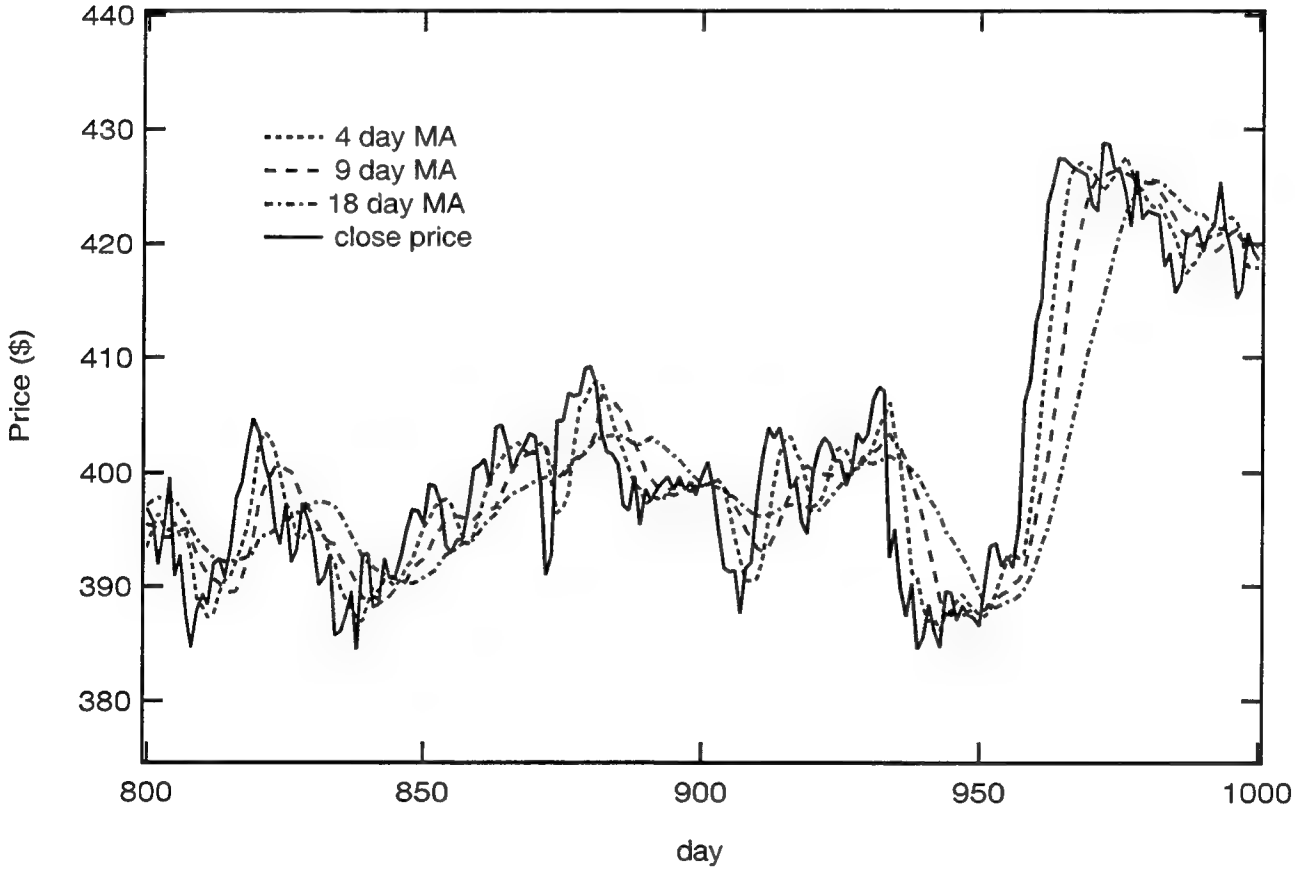

Figure 10. Moving averages of the S&P 500 futures data.

3.4.1.2 Wilder's RSI. RSI provides an indication of the overbought or oversold nature of a market. The formula for RSI is [3]:

$$RSI = 100 - \frac{100}{1 + RS},$$

where,

$$RS = \frac{ups}{downs}.$$

ups is the average of the increases in the close price in the past N days, and $downs$ is the average of the decreases in the close price in the last N days. N is the time period associated with the indicator; for a 14 day RSI, $N = 14$.

A typical RSI trading technique is to assume the market is oversold when the RSI value exceeds 70, and assume the market is overbought when the indicator drops below 30 [3]. In order to predict what is to happen tomorrow, only the current day's RSI value is required. The candidate data set retains only the 14 day RSI value for the day prior to the prediction day. Figure 11 shows the 14 day RSI indicator value for the S&P 500 data set.

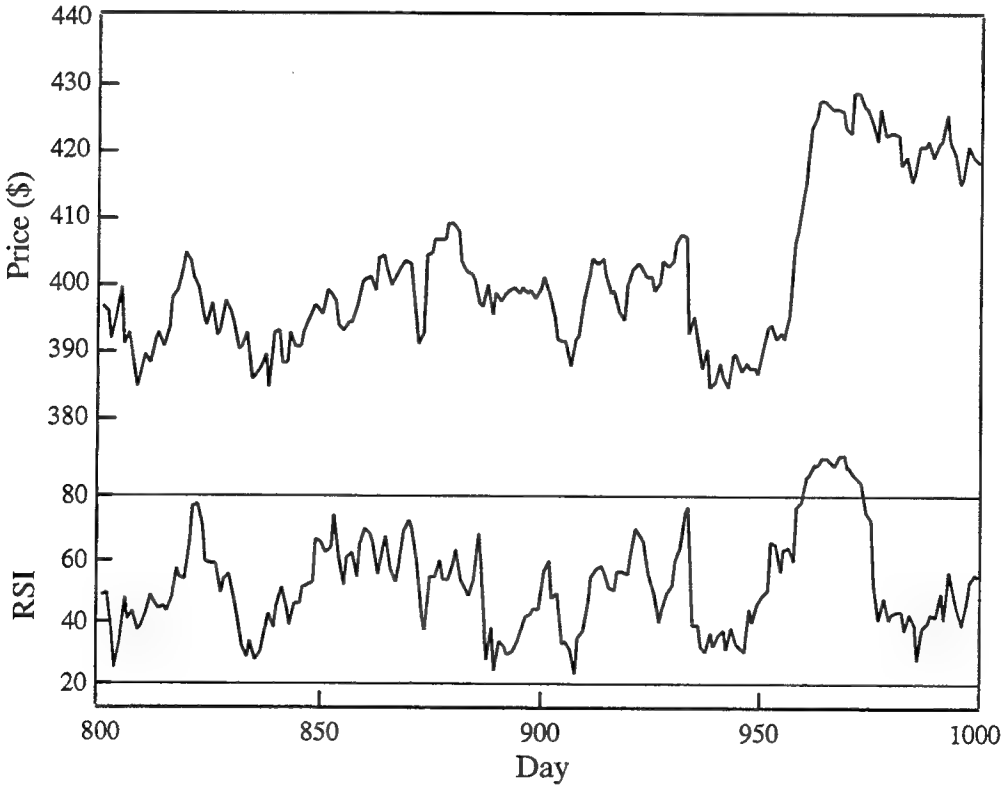

Figure 11. Wilder's RSI indicator for the S&P 500 futures data.

3.4.1.3 Stochastics. The stochastics indicator provides a measure of the position of a closing price within a time interval. The assumption is that during trending markets, prices tend to close near the range boundaries, while for nontrending markets prices tend to close near the middle of the range [3]. A stochastic indicator for a day is a pair of numbers, %K and %D. In order to compute %K and %D a raw stochastic value must be calculated. The formula for the raw value is:

$$rawvalue = 100 \times \frac{close - low(N)}{high(n) - low(N)},$$

where $high(N)$ is the highest price in the last N days, and $low(N)$ is the lowest price in the last N days. %K is a three day moving average of the raw value, and %D is a three day moving average of %K.

The stochastic indicators can be used to provide divergence information, or as overbought/oversold indicators. When the slow stochastics value's (%D) trend diverges with the current price trend, some analysts suggest this is a warning of a potential peak or trough. Stochastics can also be used in the same manner as RSI; when the indicators exceed 75 or 80 the market is said to be overbought, and when the indicator values are less than 20 to 25 the market is said to be oversold. Several of these indications are apparent in Figure 12. Since the stochastic trend information appears to be important, the stochastic indicators for the two days prior to the prediction day are retained as candidate data features.

3.4.2 Detrending and Normalization. This section describes the detrending and normalization methods used in this analysis. Time series forecasting techniques usually assume that the target time series is mean stationary [20:381-384]. This implies that no global trend is affecting the time series. Stock market data generally does not follow this assumption. Trended data can be detrended, but it is unclear how various detrending methods affect any hidden determinism in the series. Garza examined several detrending schemes in his attempts to predict the S&P 500 [11].

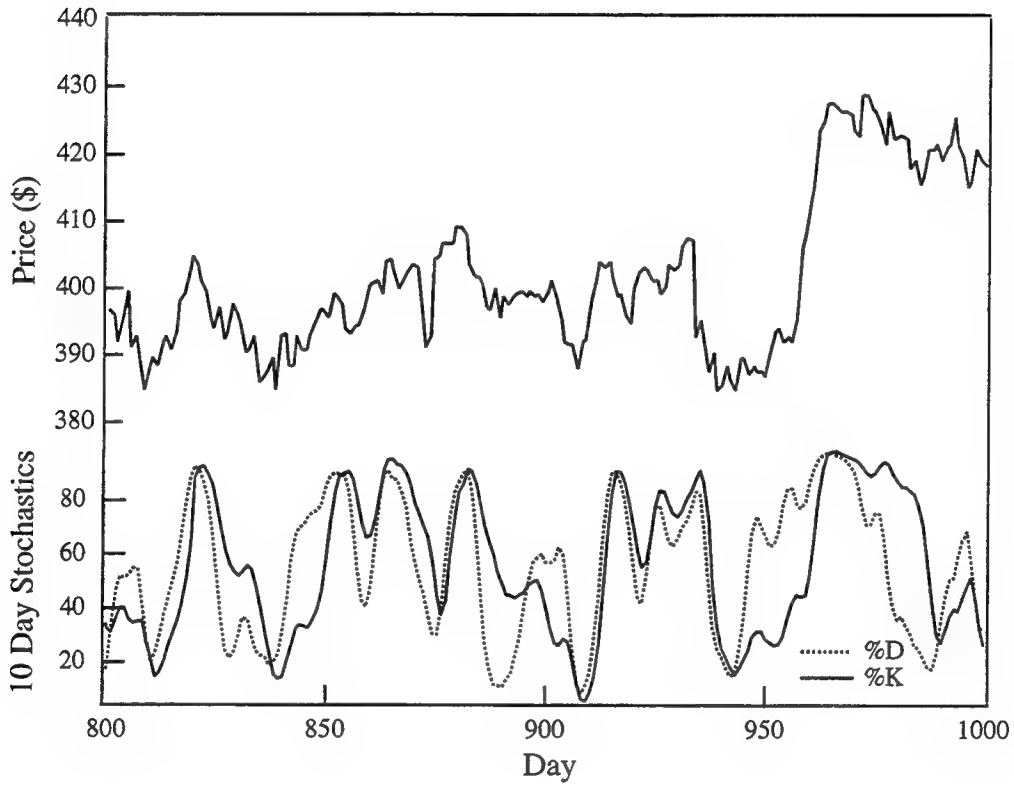

Figure 12. Ten day stochastics indicators for the S&P 500 futures data.

One method of detrending is fitting a slope and intercept to the data in a local fashion. Figure 13 shows nine time series points with a slope and intercept least squares fit to the first eight. A value for the ninth point can be estimated from this fit. The eight point window is then moved forward and a value for the tenth point can be estimated. This procedure continues until local linear estimates for the entire data set are calculated. The residuals, produced by subtracting the estimates from the true value at each point, will be stationary. Figure 14 shows the trend line and the true price for a subset of the S&P futures data. Figure 15 shows the residuals for the same subset of data. The stationarity of the residuals is apparent from the figure.

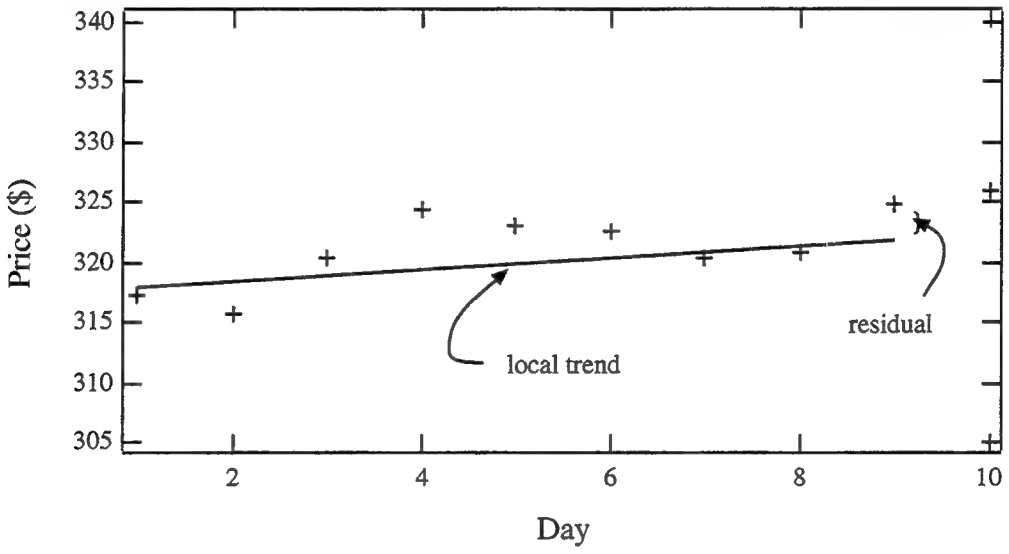

Figure 13. Local linear fit to 8 points.

In order to employ a neural network for either a classification or a regression task, the training data require some manipulation. Usually, both the input features and the target output (in the regression case) need to be reduced to a range compatible with the neural network.

The input features need to have their range limited so that the values input to the nonlinearities in the hidden layer are not in the tails of the sigmoid. The uniform random initial weight distribution will help to ensure this, but reducing the range of the input feature values to a range between -1 and 1 also aids in this process.

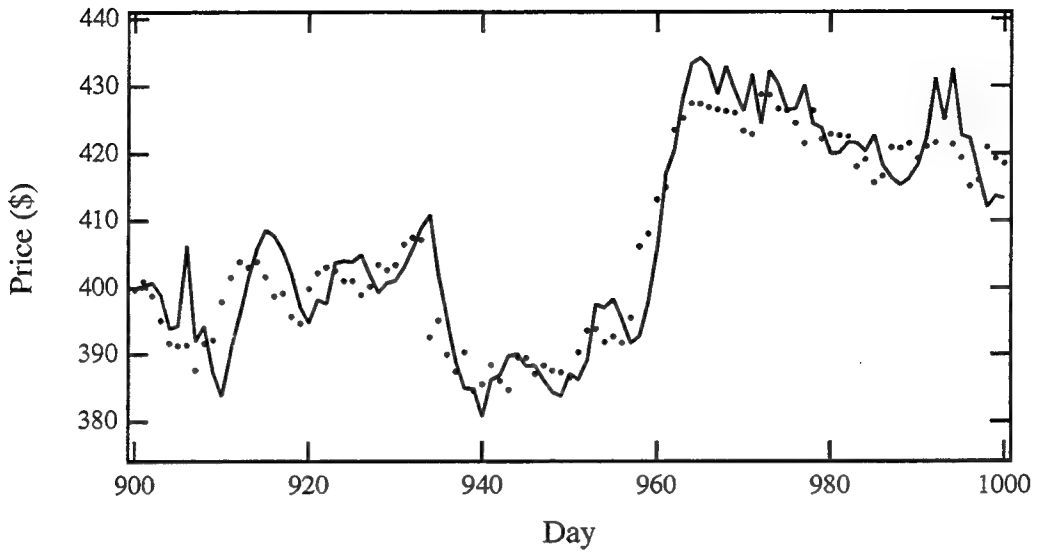

Figure 14. Comparison of the trend line and the true price data.

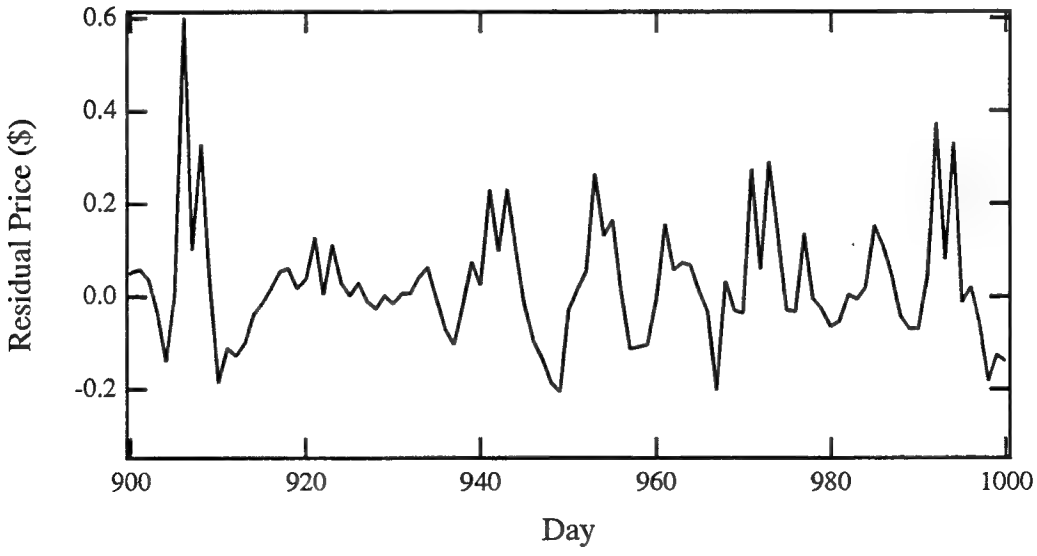

Figure 15. The residuals after fitting a local linear trend.

When using a neural network for regression or function mapping, the target output range must be reduced to a range compatible with the output sigmoid. For a symmetric sigmoid, usually a hyperbolic tangent function, the output should be between -1 and 1. These ranges are the asymptotic ranges of the sigmoids. This means that the sigmoids only reach the limits when the input to the sigmoid is infinite in either the positive or negative direction. In order to avoid forcing the sigmoid to its asymptotic value the target range is reduced even further. In this research the target range was reduced to an approximate range between -0.7 to 0.7.

The actual data set creation procedure was lengthy. First the indicators, as listed in Table 1 and defined in the previous section, were calculated. The stochastics indicators were divided by 100 for range normalization. The moving average indicators and the RSI indicator were range normalized to the range .1 to .8 by the equation:

$$xnorm = \frac{.7(x - Min(\mathbf{x}))}{Max(\mathbf{x}) - Min(\mathbf{x})} + .1, \tag{14}$$

where $Max(\mathbf{x})$ and $Min(\mathbf{x})$ are the maximum and minimum of the training portion of the indicator feature. The training set size used was 800, and the test set used was the next 300 points. To calculate a normalized RSI value, for instance,the maximum and minimum of the first 800 data points are used. The eight stochastics data vectors, and the time series data vectors were then appended to the normalized moving average and RSI data vectors.

The resulting data set consists of an 1100 by 23 matrix. Table 2 lists the matrix columns. The notation *t, t-1, t-2* refers to the target day, and one and two days prior respectively. Another data set with normalized raw price data was also created to determine the benefit of detrending. Column 21 is the price we are trying to predict with the other 20 input features. Any feature subsets to be evaluated were extracted from this master data matrix using *Awk* (see Appendix B).

Column	Indicator	Day
1	RSI	(t-1)
2	4-day Moving Average	(t-2)
3	4-day Moving Average	(t-1)
4	9-day Moving Average	(t-2)
5	9-day Moving Average	(t-1)
6	18-day Moving Average	(t-2)
7	18-day Moving Average	(t-1)
9	10-day %K	(t-2)
9	10-day %K	(t-1)
10	10-day %D	(t-2)
11	10-day %D	(t-1)
12	25-day %K	(t-2)
13	25-day %K	(t-1)
14	25-day %D	(t-2)
15	25-day %D	(t-1)
16	Close Price	(t-5)
17	Close Price	(t-4)
18	Close Price	(t-3)
19	Close Price	(t-2)
20	Close Price	(t-1)
21	Close Price	(t)

Table 2. The columns of the data matrix

3.5 Summary

This chapter had two goals, (1) to explain contributions to Lee and Landgrebe's work, and (2) to present the data preparation and methodology for the experiments conducted on the stock market data. Lee and Landgrebe's basic procedure was applied to an artificial neural network classifier, and a simple example was used to explain the technique. A saliency metric based on the contribution of the input features to the discriminantly informative direction was developed. The Lee and Landgrebe technique, and the saliency metric were then modified for feature extraction without a classifier. Finally a brief justification for using the Lee and Landgrebe technique for regression is presented.

The second part of this chapter explained the data preprocessing applied to the S&P 500 data. First, definitions and brief explanations of candidate indicators are provided. Then detrending to create a stationary data series is discussed. A new local linear detrend method is explained, and the normalization of the input and output data is discussed.

The next chapter presents the results of the application of these techniques to the S&P 500 data set.

IV. Results and Discussion

4.1 Introduction

Chapter III covered the development of saliency tools, and the data preprocessing of the S&P 500 data set. This chapter presents the results of the application of these tools. First the local-linear detrend forecasting results are compared with results for a data set that is not detrended. The next section takes a basic enumeration approach to determining which of the indicators is most useful for improving neural network prediction. The final section presents saliency results from a Ruck type saliency metric and the proposed metric based on decision boundaries.

Just getting a neural network to learn is not always a simple task. The approach applied here was to start with too many middle nodes and reduce the number of middle nodes until the test set error was reasonably close to the training set error. This approach was chosen to allow determination of the other network parameters, step size and momentum, with a network that was certain to have the capability to model the complexity of the data. Comparisons of symmetric versus logistic output sigmoids were made and two methods of presentation were attempted. This preliminary analysis was performed on only the close price values and the target values (rows 16 to 21 of Table 2) to provide a baseline set of prediction results. The final architecture and parameters are presented in Table 3.

Parameter	Value or Type
Number of hidden nodes	10
Type of output sigmoid	Symmetric
Stepsize (η)	0.01
Momentum (α)	0.8
Number of epochs	12000
Presentation	Random

Table 3. The baseline network parameters.

4.2 Comparison of Local-Linear Detrend to Raw Data

In order to compare results of various runs a metric is required. The neural net software provides root mean square error (rmse), so this was chosen. The best results obtained on the local linear detrended baseline data set resulted in an rmse of 0.132 for the 800 point training set and 0.142 for the 300 point test set. In this case, the net is predicting the difference between the local linear detrend and the actual value. These residuals appear to be randomly distributed about zero. The rmse of the residuals without any attempt at modeling is 0.176 for the training set and 0.162 for the testing set. The neural network is able to find some determinism in what appears in Figure 16 to be random data. The impact of this reduction in rmse is evident in Figures 17 and 18 where the original data, the trendline and the results of training the network are depicted.

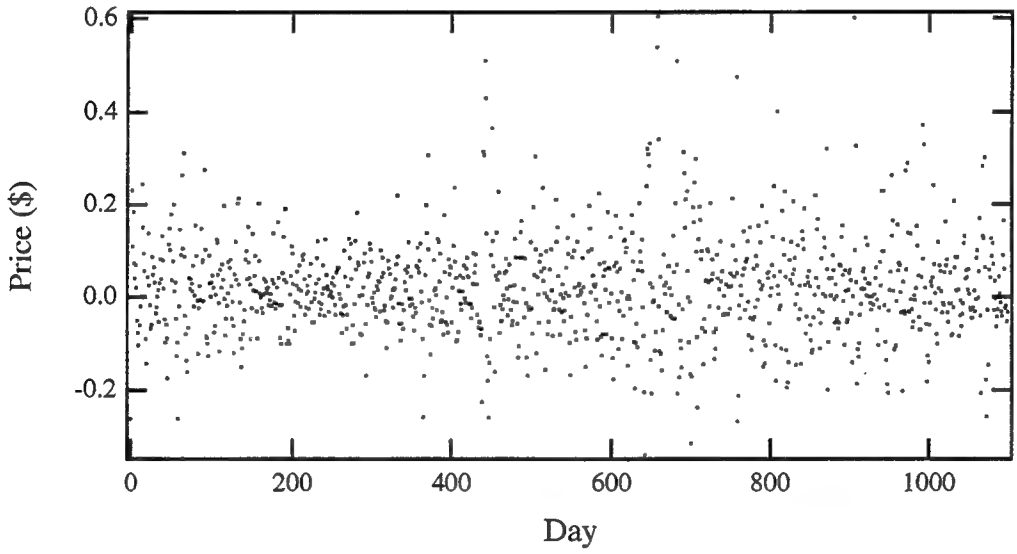

Figure 16. Scatter plot of residuals after linear detrend.

To enable comparison of the detrend results to the non-detrended results, the actual raw rmse is used. The raw rmse is the root mean square error of the retrended denormalized network results, in the detrended case. In the non-detrend case the raw rmse is the root mean square error of the denormalized network results. The network used to fit the non-detrended data differed from the baseline network in that it was trained for only 2000 epochs. The

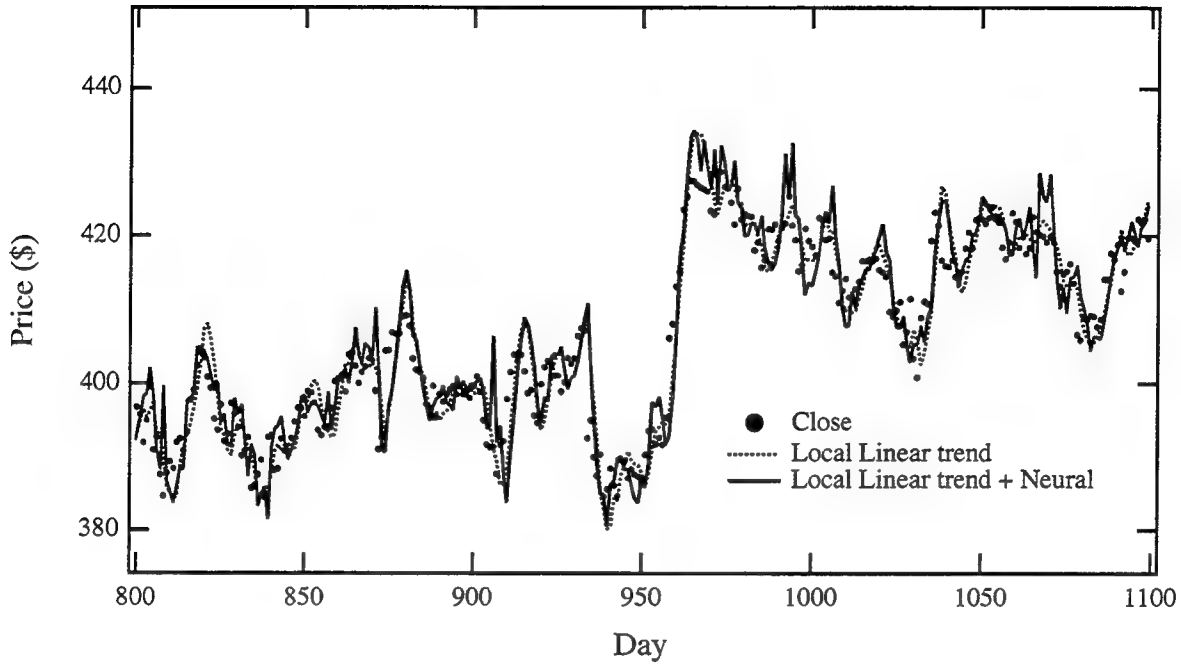

Figure 17. Comparison of true price, trend only and trend with net for baseline data.

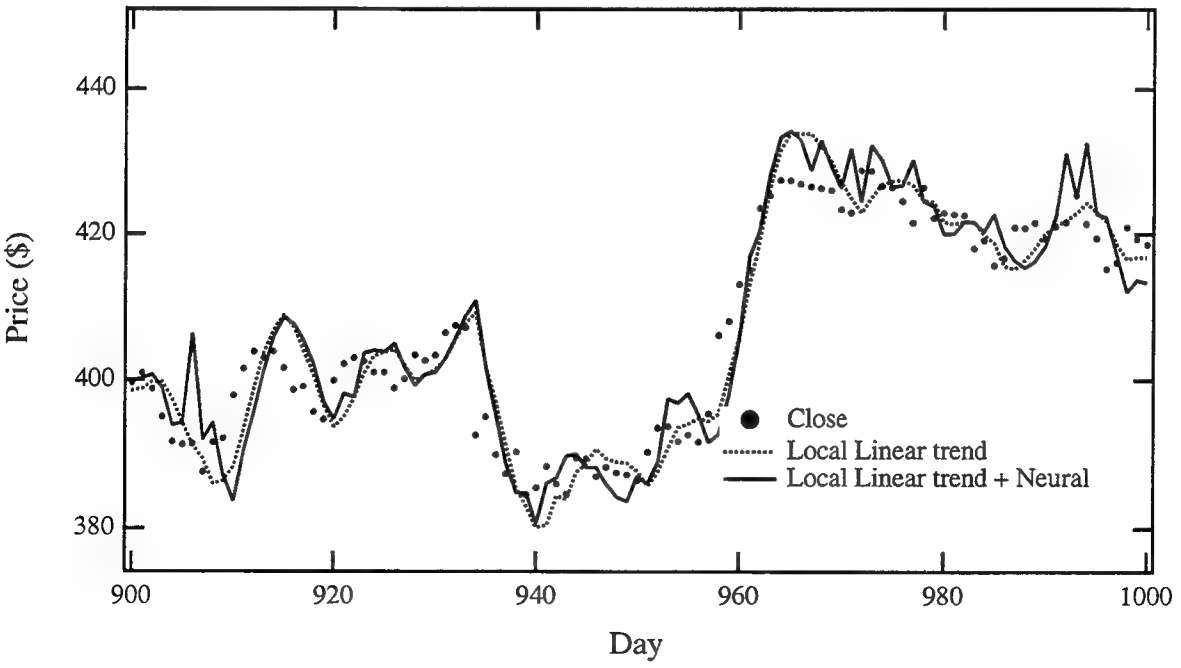

Figure 18. Baseline for days 900 to 1000.

Detrending	test set rmse
Local Linear	4.73
None	6.75

Table 4. The effect of detrending.

training error appeared to have stabilized at this point. The raw rmse results are presented in Table 4. Figures 19 and 20 show the first two hundred points of the test set with both the detrended and fit results and the non-detrended and fit results. It is not apparent from this figure which is better. The non-detrended results seem to be biased low after the data take the step on day 958. This could be the result of not detrending. However, the range normalization for the non-detrended data was based on the first 800 points, so the output sigmoid is being forced closer to the asymptote. The maximum target value is 0.921. Based on the rmse results the rest of this work uses only the local linear detrended data.

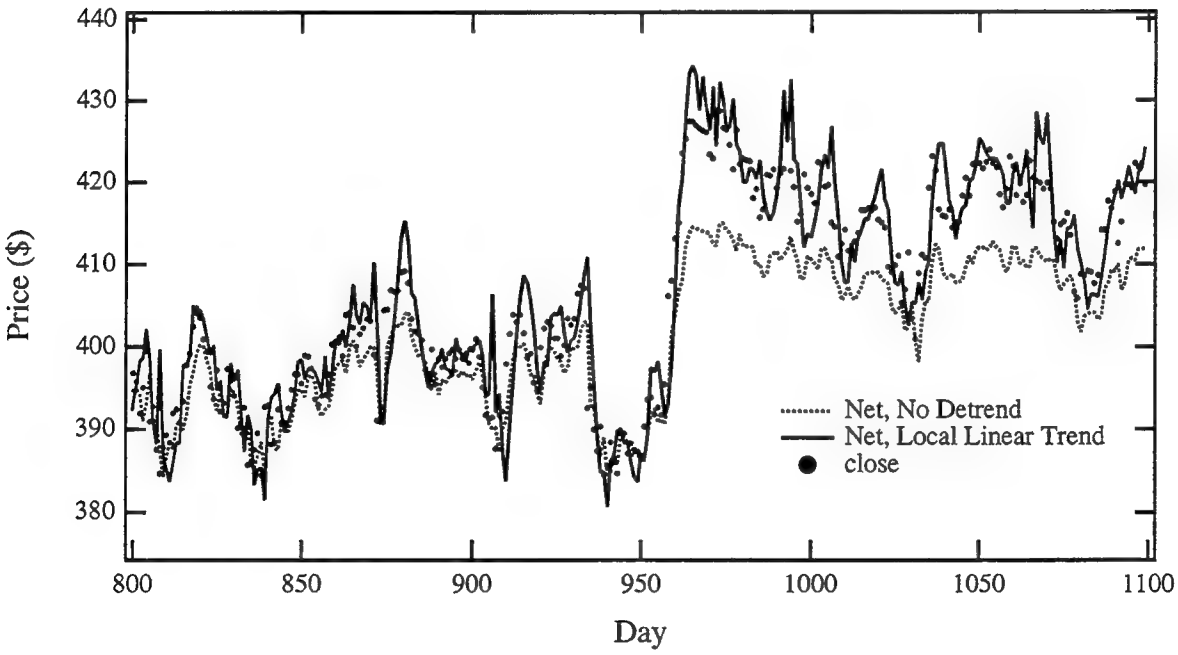

Figure 19. Comparison of local linear detrending and no detrending.

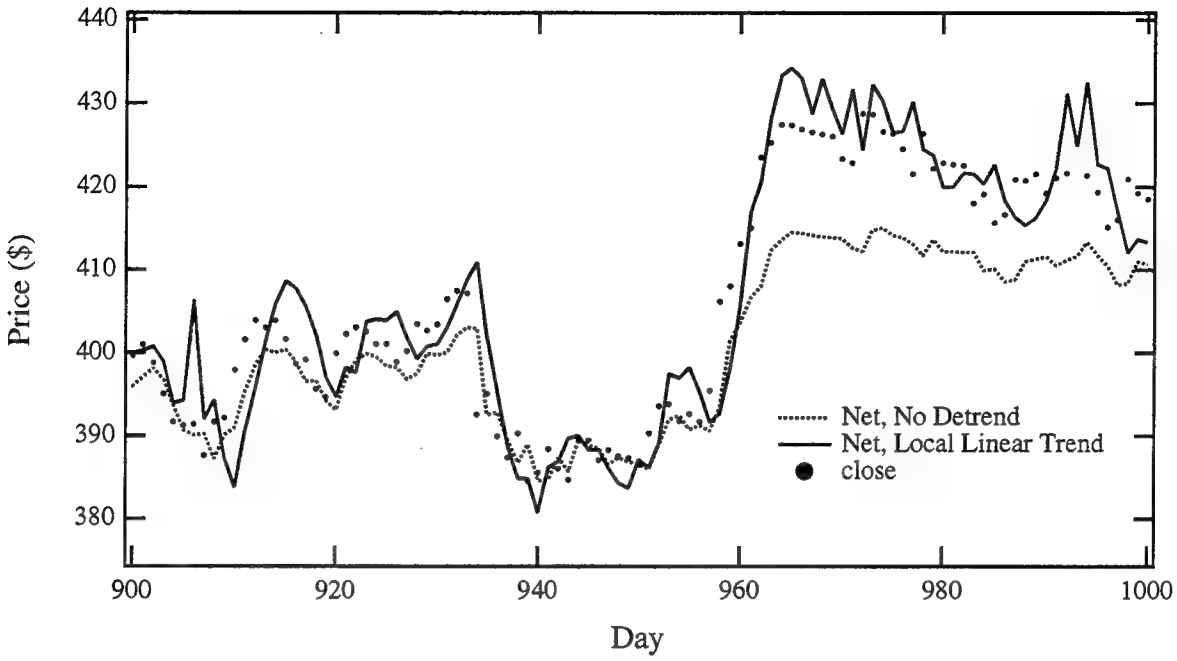

Figure 20. Detail of comparison of local linear detrending and no detrending.

4.3 Partial Enumeration

Even with just 6 candidate feature subsets, 64 computer runs are required for complete enumeration of all possible combinations of feature subsets. In addition, for each additional random seed (for the initial weights) evaluated, 64 more runs are added. The approach taken here is to enumerate the indicator subsets choosing one, two and possibly three indicator subsets. When additional features are added the net parameters are no longer optimum. Retuning for each feature subset is avoided here to ensure reported rmse results are due to the feature subsets and not the net tuning. If overparameterization, indicated by a large test set rmse relative to the training set rmse occurs, hidden nodes may be removed on a case by case basis. Similarly, if the net fails to learn, hidden nodes may be added, again on a case by case basis. Table 5 displays the mnemonics to be used for each indicator subset. Table 6 shows the results of evaluating the baseline data set augmented with each indicator subset.

An initial tentative conclusion about the usefulness of the moving average data to the neural net data can be drawn. For each moving average testing result the rmse exceeds the

Mnemonic	Indicator	Number of associated features
r	RSI	1
m4	4-day Moving Average	2
m9	9-day Moving Average	2
m18	18-day Moving Average	2
s10	10-day Stochastics	4
s25	25-day Stochastics	4

Table 5. Mnemonics for indicator feature subsets.

Feature Subset	Seed 1 rmse		Seed 2 rmse	
	Train	Test	Train	Test
r	0.127	0.129	0.129	0.135
m4	0.123	0.436	0.165	0.523
m9	0.125	0.346	0.164	0.376
m18	0.128	0.208	0.162	0.213
s10	0.124	0.140	0.123	0.130
s25	0.124	0.129	0.124	0.125

Table 6. Network results after adding one indicator feature subset.

detrended residuals rmse (0.162). Over parameterization may explain this result. If an input provides no independent information, yet increases the number of weights or parameters in a neural net, the network is more likely to memorize the training set. This argument seems reasonable for the 4-day moving average, since the network has all the information it needs to create its own 4 day moving average. Unfortunately its not so reasonable for the 9 and 18 day moving averages.

To gain more insight into this over parameterization issue the moving averages were evaluated with a reduced number of hidden nodes. Recall that the baseline number of hidden nodes is ten. Networks with eight and nine hidden nodes were trained and the results are presented in Tables 7, and 8. Little improvement is seen with the 4-day and 9-day moving averages, but the 18-day moving averages perform significantly better with reduced hidden nodes.

Feature Subset	Seed 1 rmse		Seed 2 rmse	
	Train	Test	Train	Test
m4	0.123	0.458	0.124	0.488
m9	0.125	0.379	0.124	0.150
m18	0.126	0.158	0.129	0.137

Table 7. Moving average results for nine hidden nodes.

Feature Subset	Seed 1 rmse		Seed 2 rmse	
	Train	Test	Train	Test
m4	0.130	0.474	0.126	0.490
m9	0.130	0.327	0.127	0.208
m18	0.132	0.158	0.131	0.135

Table 8. Moving average results for eight hidden nodes.

The next step in this enumeration procedure is to choose pairwise indicator feature subsets. Table 9 presents the results of these calculation. Very few of the test results are better than the detrended residual rmse (0.162). In fact only three of the pairwise subsets are superior to this value. The combinations of RSI and each of the stochastics indicators and the combination of the two stochastics indicators do relatively well, again suggesting that the moving averages add very little.

As the testing errors are increasing, the networks should be retuned. Retuning now involves optimizing 15 networks. Instead of attempting to tune each net, one more step in the enumeration process is taken, evaluating the indicator feature subsets three at a time. In this case only four of the test results show improvement over the residuals themselves. The first of these is RSI with 4 and 9 day moving averages. The second is RSI, with 18 day moving averages and 10 day stochastics. The third is RSI with 10 and 25 day stochastics. This is the only test case where both random seeds provided results better than the residuals. The final successful case is 10 day stochastics with 4 and 9 day moving averages.

At this point proceeding further with enumeration technique without retuning would be useless, so the experiment was terminated.

Feature Subset	Seed 1 rmse		Seed 2 rmse	
	Train	Test	Train	Test
r, m4	0.122	0.487	0.117	0.542
r, m9	0.124	0.190	0.120	0.517
r, m18	0.122	0.462	0.120	0.475
r, s10	0.120	0.138	0.128	0.140
r, s25	0.122	0.127	0.131	0.147
m4, m9	0.122	0.394	0.120	0.430
m4, m18	0.122	0.474	0.121	0.487
m4, s10	0.116	0.442	0.164	0.376
m4, s25	0.117	0.307	0.116	0.442
m9, m18	0.124	0.274	0.124	0.321
m9, s10	0.116	0.448	0.115	0.421
m9, s25	0.117	0.363	0.118	0.345
m18, s10	0.116	0.356	0.112	0.456
m18, s25	0.118	0.256	0.117	0.391
s10, s25	0.118	0.135	0.119	0.140

Table 9. Network results after adding two indicator feature subsets.

In summary, the enumeration approach without retuning has suggested that some of the indicator feature subsets are apparently more useful to the neural net than others. The single and pairwise indicator experiments suggest that RSI, and the two stochastics indicators are most useful. The three indicator experiments suggest that moving averages may be slightly useful, but the most consistent results are obtained with the RSI and stochastics indicators. The next section applies individual saliency methods to the indicators.

4.4 Individual Saliencies

The enumeration approach of the previous section allows determination of the impact of indicator feature subsets on the neural network's ability to model the data. In this section individual input features are compared. The ten day stochastics indicator feature subset, for instance, encompasses four individual input features. The candidate data set for this saliency analysis is chosen from the three indicator experiments in the Partial Enumeration section. The three indicators RSI, 10 and 25 day stochastics as well as the 5 previous days close price

Feature Subset	Seed 1 rmse		Seed 2 rmse	
	Train	Test	Train	Test
r, m4, m9	0.117	0.515	0.119	0.521
r, m4, m18	0.120	0.507	0.123	0.446
r, m4, s10	0.117	0.151	0.116	0.391
r, m4, s25	0.119	0.185	0.112	0.446
r, m9, m18	0.123	0.508	0.130	0.477
r, m9, s10	0.114	0.266	0.119	0.193
r, m9, s25	0.117	0.332	0.113	0.438
r, m18, s10	0.118	0.154	0.117	0.432
r, m18, s25	0.116	0.343	0.114	0.399
r, s10, s25	0.126	0.155	0.116	0.151
m4, m9, m18	0.121	0.407	0.121	0.215
m4, m9, s10	0.114	0.360	0.117	0.153
m4, m9, s25	0.118	0.262	0.121	0.177
m4, m18, s10	0.117	0.377	0.117	0.265
m4, m18, s25	0.115	0.365	0.123	0.171
m4, s10, s25	0.119	0.349	0.114	0.422
m9, m18, s10	0.120	0.282	0.113	0.342
m9, m18, s25	0.118	0.257	0.122	0.170
m9, s10, s25	0.116	0.334	0.114	0.408
m18, s10, s25	0.115	0.383	0.112	0.400

Table 10. Network results after adding three indicator feature subsets.

Column	Indicator	Day
1	RSI	(t-1)
2	10-day %K	(t-2)
3	10-day %K	(t-1)
4	10-day %D	(t-2)
5	10-day %D	(t-1)
6	25-day %K	(t-2)
7	25-day %K	(t-1)
8	25-day %D	(t-2)
9	25-day %D	(t-1)
10	Close Price	(t-5)
11	Close Price	(t-4)
12	Close Price	(t-3)
13	Close Price	(t-2)
14	Close Price	(t-1)
15	Close Price	(t)

Table 11. The columns of the three indicator data matrix

seemed to provide the most consistent results in the three indicator enumeration results. This indicator feature set results in a data set with 14 input features and 1 output feature. The features and their columns are displayed in Table 11.

The accuracy of any derivative based saliency depends on the range of the input feature. As both the decision boundary based metric and Ruck's metric depend on the partial derivatives of the output with respect to the input, it is important that the input features have consistent ranges to ensure a fair comparison of the partial derivatives. With this in mind, the input features were renormalized to have a range of one. This presented a problem for the close price data (columns 10 through 14) since one significant outlier caused considerable skewing of the data. Two possible solutions are (1) to use a mean, standard deviation based normalization or (2) to ignore the one outlier. The latter solution was chosen here. The results of five runs of the renormalized data, with each run having a different seed are shown in Table 12.

4.4.1 Ruck Metric. The first individual saliency metric evaluated is the Ruck metric, or the sum of the partial derivatives of each input feature over the input space. The results for

Run	Train	Test
1	0.119	0.153
2	0.118	0.142
3	0.115	0.164
4	0.120	0.154
5	0.143	0.170

Table 12. Network results after renormalizing RSI, 10 and 25 day stochastics.

Feature	Ruck Saliency Metric				
	Run 1	Run 2	Run 3	Run 4	Run 5
1	0.484056	0.540412	0.617555	0.342015	0.344853
2	0.541894	0.412844	0.558808	0.522800	0.421346
3	0.827932	0.857118	0.389490	0.894576	0.544237
4	0.960617	0.510921	0.495022	0.587752	0.581671
5	0.652345	0.542558	0.539575	0.814176	0.680770
6	0.512810	0.341850	0.517583	0.360040	0.342596
7	0.487779	0.590069	0.214170	0.514803	0.254195
8	0.577659	0.563539	0.810572	0.271782	0.315668
9	0.859564	0.602370	0.534907	0.439418	0.216937
10	0.366554	0.472997	0.401336	0.355781	0.347492
11	0.334688	0.311575	0.326111	0.325615	0.356960
12	0.342520	0.365239	0.485877	0.348681	0.271373
13	0.341517	0.319410	0.405947	0.322356	0.346950
14	1.000000	1.000000	1.000000	1.000000	1.000000

Table 13. Ruck saliency results for each run.

each run are shown in Table 13. The Ruck metric finds feature 14, the previous days closing price (after detrending and normalization) to be the most salient in all the runs. Comparing saliencies for the best run, Run 1, with the worst, Run 6 provides insight into which indicators seem to be responsible for producing the good results. In the best case, Run 1, the 10-day %K value is only slightly less salient than the previous day closing price. The saliencies for Run 5 rank this indicator as fourth. The 25-day %D is third in saliency for Run 2 and last in saliency for Run 5. Further analysis along these lines seems to show that Run 2 makes more use of the previous day stochastics indicators than Run 5.

Feature	Decision Boundary based Saliency Metric				
	Run 1	Run 2	Run 3	Run 4	Run 5
1	0.049543	0.051400	0.149099	0.031521	0.049107
2	0.101837	0.060858	0.118985	0.059351	0.076839
3	1.000000	0.875852	0.168805	1.000000	1.000000
4	0.287553	0.208087	0.170760	0.291298	0.170151
5	0.113617	0.134965	0.140198	0.170240	0.290087
6	0.064311	0.121978	0.130076	0.046619	0.149710
7	0.103994	0.136429	0.015819	0.061524	0.023240
8	0.199970	0.272866	0.381245	0.014450	0.105731
9	0.231607	0.141965	0.154730	0.072351	0.048042
10	0.044744	0.087374	0.108044	0.038351	0.054698
11	0.052575	0.022241	0.040724	0.129154	0.181723
12	0.053872	0.045879	0.133302	0.062130	0.043966
13	0.106134	0.090228	0.082366	0.095699	0.155992
14	0.712340	1.000000	1.000000	0.951859	0.838163

Table 14. Decision boundary based saliency results for each run.

4.4.2 Decision Boundary Based Saliency Metric. The decision boundary based saliency metric results are in Table 14. Some differences from the Ruck metric are immediately apparent. First, only two of the five result in the previous day close price being most salient. For the other three runs, feature 3, the 10-day %K value for the previous day, is the most salient.

4.4.3 No-Classifier Saliency Metric. The no-classifier decision boundary approach explained in Chapter II was applied to the renormalized training data. The lengths of the pseudo-normals were not scaled to unit length for this experiment. The results are shown in Table 15. The rankings do not seem to agree with either the Ruck or the decision boundary based metric. The probable cause of this poor performance is the overlapped nature of the data, and the resulting diffusion of relevant discriminant directions with the directions caused by outliers. The section on Lee and Landgrebe without a classifier in Chapter III addressed this problem.

Feature	Saliency	Rank
1	0.778308	4
2	0.503643	8
3	0.290952	12
4	0.406354	9
5	0.547180	7
6	0.343823	11
7	0.215898	14
8	0.273473	13
9	0.355708	10
10	1.000000	1
11	0.849615	3
12	0.687976	6
13	0.690607	5
14	0.859435	2

Table 15. No classifier decision boundary based saliency results.

4.4.4 Summary and Comparison of Saliency Rankings. A comparison between the average over the five runs of the two metrics for each feature is shown in Table 16. This table also shows the average saliency ranking of each feature for each metric, including the no-classifier metric.

4.4.5 Discussion. Both the Ruck and the decision based saliency methods agree that the six most salient features are columns 3, 4, 5, 8, 9 and 14. The features associated with these columns are shown in Table 11. After these six features, the two methods diverge in their choice of features. Some of these results are expected. For instance, it seems reasonable that the previous day's price value is important in predicting the current day's price. The absence of any of the other price features from the top six is interesting. The RSI indicator seems to have mixed results. The decision boundary based saliency ranks RSI dead last, while Ruck's saliency ranks RSI as eighth. The stochastics indicators, and the slow stochastics (%D) features in particular, seem to be relevant, while only one fast stochastic (10 day %K) feature appears in the top six.

Feature	DB Sal.	Ruck Sal.	DB Rank	Ruck Rank	NC Rank
1	0.066134	0.465778	14	8	4
2	0.083574	0.491539	10	7	8
3	0.808931	0.702671	2	2	12
4	0.225570	0.627196	3	4	9
5	0.169821	0.645885	5	3	7
6	0.102539	0.414976	8	9	11
7	0.068201	0.412203	11	10	14
8	0.194853	0.507844	4	6	13
9	0.129739	0.530639	6	5	10
10	0.066642	0.388832	13	11	1
11	0.085283	0.330990	9	14	3
12	0.067829	0.362738	12	12	6
13	0.106084	0.347236	7	13	5
14	0.900472	1.000000	1	1	2

Table 16. Comparison of decision boundary based (DB) saliency and Ruck saliency averaged over 5 runs, along with no classifier (NC) rankings.

The no-classifier rankings do not agree with either of the other metric rankings. The most salient no-classifier feature is ranked eleven and thirteen by the other metrics.

The ultimate test of these saliency techniques is the prediction improvement they provide. A brief attempt was made to determine if the salient features provided better prediction results than the full data set. The nine most salient features as determined by the decision boundary based metric were used to predict the closing price. Although the improvement in rmse was not very large, the net trained quicker, and the test results were actually better than the training results, a reasonable result since the test set rmse was smaller than the training set rmse. After training for 10000 epochs, the training set rmse was 0.134 and the test set rmse was 0.123. These rmse's are not much better than the 25 day stochastics one indicator feature subset (train 0.124, test 0.125), however the net trained quicker. Figures 21 and 22 graphically depict the results. Comparison with the baseline case, Figures 17 and 18, shows the importance of the slight improvement in rmse. For example, the fit line for the best results appears much less jagged than the baseline case.

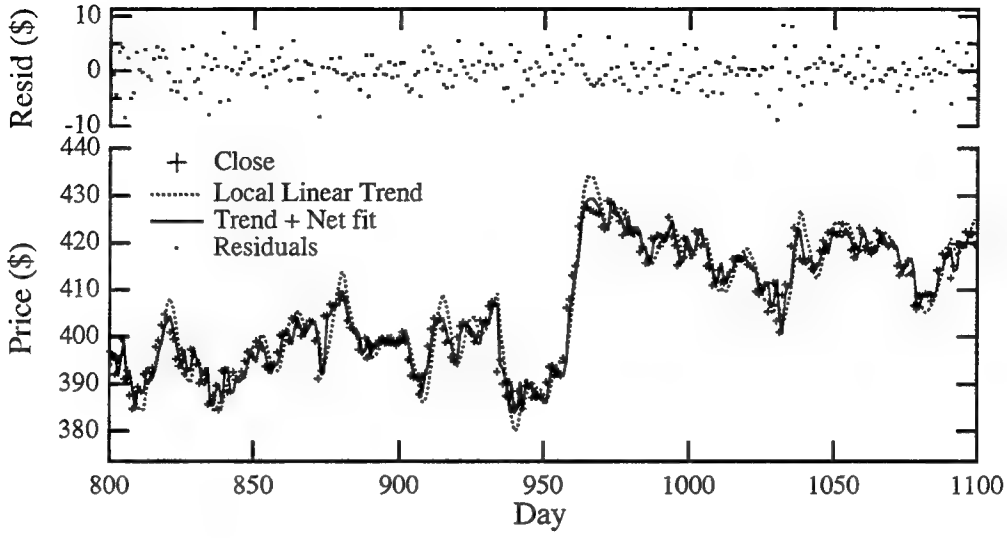

Figure 21. Comparison of true price, trend only and trend with net for best results obtained.

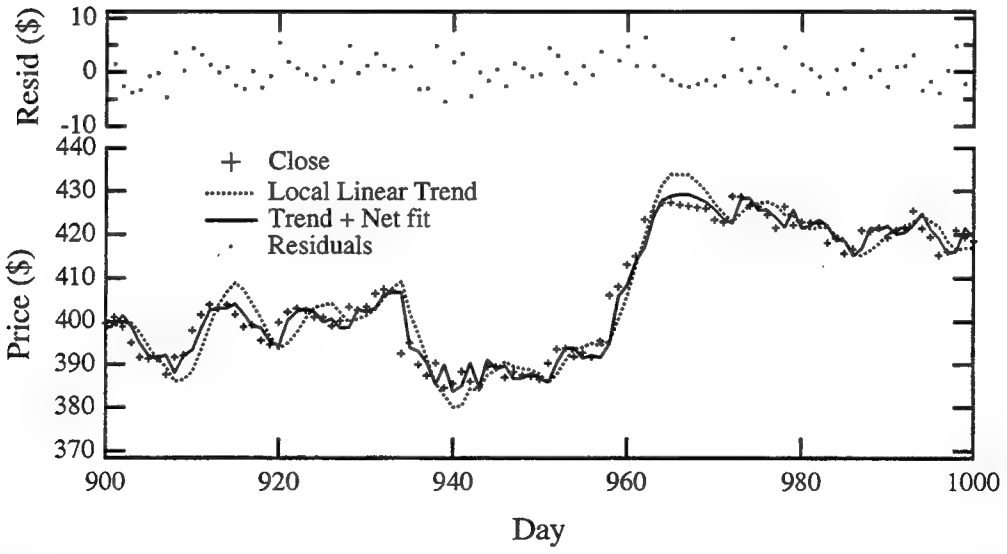

Figure 22. Comparison of true price, trend only and trend with net for best results obtained, expanded to show detail.

Other S&P 500 futures prediction research, including [11, 2], use the trend accuracy as a fit metric. The trend accuracy is simply the ratio of the number of times the prediction scheme predicted the correct movement direction for the data set. With the S&P 500 futures data this corresponds to up and down price movements. Using the best results from the nine most salient features, the trend calculation accuracy was 52.0%. This is comparable to Azoff's result of 53.9% trend accuracy, although he is using other indicator information and more sophisticated neural networks [2:116]. Garza managed to average 56.50% accuracy using a hybrid method which utilized both a neural network and DVS techniques [11:72].

The decision to use a regression approach as opposed to a trend classification approach was based on the importance of accurately predicting large movements. If the small price changes, between -3.00 and 3.00, are discounted the nine feature saliency results in 59.27% accuracy.

4.5 Summary

This chapter presents the results of application of neural network saliency techniques to the S&P 500 futures data. First a comparison of results obtained with a new local linear detrending with non-detrended data set is presented. The results are mixed. Although the local linear method is superior in terms of test set rmse, this could be due to the normalization of the non-detrended data set. The decision is made to continue with the local linear detrend data set.

Partial enumeration of indicator features subsets is attempted in the second section. The results of training a baseline network with one, two and three indicator subsets are provided. RSI and stochastics indicators seem to be the most valuable in terms of reduction of test set root mean square error.

The last section presents the results of applying the Ruck and decision boundary based individual saliency metrics to 14 individual features including the RSI, 10 day stochastics and 25 day stochastics subsets. The two metrics seem to agree on the six most salient features.

This top six feature set is composed of the four %D values, the previous day's price and one 10 day %K statistic. A brief attempt is made to utilize the saliency metrics to improve prediction.

The next chapter presents some of the conclusions drawn from these results and presents recommendations for further research.

V. Conclusion and Recommendations

5.1 Introduction

The purpose of this research is to examine ways to improve nonlinear time series through the use of additional time series. The additional time series considered here are derived from the target time series. Three methods for determining the usefulness of the non-target time series are applied to S&P 500 futures data and indicators. These methods include partial enumeration, Ruck's saliency metric and a decision based saliency metric derived in this research. An additional classifier independent saliency metric is designed and evaluated. Chapter IV presents the results of the application of the three saliency methods to the S&P 500 data. This chapter will provide a summary of the results of this research, and draw some general conclusions based on these results. A list of contributions to the fields of neural network based saliency calculations and S&P 500 futures predictions concludes this chapter.

5.2 Summary and Discussion of Results

Partial enumeration of indicator subsets provided a useful notion of which of the indicator feature subsets are most influential in improving a neural network based prediction model. Although no claims of statistical significance can be made using just two runs per subset, a general feel for the importance of the indicators is provided. Even with more runs it is not clear that statistical significance can be determined. The problem lies in the dependence of the network on optimization. One can quote results from a network with ten hidden nodes but the question of whether ten hidden nodes is the correct number of hidden nodes for all the subsets remains unanswered. Steppe's architecture selection methodology provides one solution to this problem [27]. Even the Steppe approach, however, assumes that a net that is optimized for the full feature set (in terms of momentum, step size and number of training epochs) will be optimal for feature subsets. The impact of tuning or optimizing a network needs to be taken into account.

The Ruck saliency metric was applied primarily to provide a comparison with the derived decision boundary metric. The decision boundary based metric provides similar results to the Ruck saliency with fewer partial derivative calculations. The attempt to use a reduced set of salient features provided good results. Here again, the question of network optimization arises. The saliency metric is obtained from the best of the partial enumeration runs, which are not optimized. In order to use a smaller set of salient features for classification the net must be retuned. Are the good salient subset results due to the quality of the features selected, or merely an artifact of the tuning? The only possible indication that the features themselves are responsible is the relative change in training versus test error. In particular, all training accomplished on the salient features resulted in test set errors that were smaller than the training set errors. Increasing the number of hidden nodes by 50% did not affect this result. All the baseline and enumeration runs resulted in test set errors that were larger than the training set error. In the moving average case, even after reducing the number of hidden nodes, the test set error was larger than the rmse of the raw target data. This provides some indication of the robustness of the salient feature subset model.

5.3 General Conclusion

The results of this research indicate that additional time series data can be used to increase the predictive capability of a neural network on a nonlinear time series. The feature selection techniques evaluated provided a 13.4% reduction in root mean square error. This calculation is based on the difference between the baseline S&P 500 at the beginning of Chapter IV and the saliency chosen indicator feature subset discussed at the end of Chapter IV.

5.4 Contributions

In this research decision boundary based feature extraction techniques are extended for use in neural networks. Based on these extensions, a new decision boundary based saliency metric is defined and compared to previous metrics. A new classifier independent feature

extraction/saliency method is defined and explored with mixed success. The potential for extracting features without regard to classifier deserves more research. A set of relevant indicators to the prediction of the S&P 500 futures price is determined. Finally, the importance of indicators such as RSI and stochastics over moving average indicators in neural net prediction of the financial data was discovered through partial enumeration of indicator subsets.

Appendix A. Lee and Landgrebe for Neural Networks

A.1 Introduction

This appendix demonstrates the use of *Mathematica* to perform decision boundary analysis on a neural network based classifier. Any functions that are not intrinsic to *Mathematica* are included in listings at the end of this appendix.

A.2 Discriminant Function

After successfully training a network to classify the data, the weights are extracted from the LNKmap .param file. The .param file is a text file, and the weights are easily located. The LNKmap weight .param file includes a listing for the weights from the input layer to the hidden layer bias, listed as the last *number of input nodes* weights in the first weight matrix. These must be ignored when extracting the weights. The data for this example consisted of 298 vectors of 3 features and a target. The network had 3 input nodes, 6 hidden nodes and 1 output node.

Read the data from the same files used to train and test the neural network. The first argument of ReadList is a filename.

```
train = ReadList[
  "Socrates:thesis:Experiments:sinwnoise:sin.train",
Number,RecordLists->True];
test = ReadList[
  "Socrates:thesis:Experiments:sinwnoise:sin.test",
Number,RecordLists->True];
output=Join[train,test];
Length[output]}

298
```

Extract the weights from the LNKmap .param file.

```
{w1,w2} = {{
{0.0992727, 1.03985, -1.86179, 0.0583072},
```

```
{0.078818, -0.732222, 1.61733, -0.238891},
{0.0430646, -0.147562, -0.581437, -0.126756},
{0.0655318, 0.555042, -1.42235, -0.0519397},
{-0.020631, -0.013815, 0.498293, -0.0191677},
{0.0994582, -0.823559, 1.69069, -0.188408}},
{-2.92588, 2.45208, -0.641258, -1.95943,
 0.532183, 2.6143, 0.20734}};
```

Build the mapping function. These are the hidden and ouput sigmoid functions, respectively.

```
sigmoid[x_] := 1/(1+E^-x)//N
sigmoido[x_] := (2/(1+ E^-x) -1)//N
```

This is the feedforward network regression function.

```
h[x_] :=
sigmoido[(w2.Join[sigmoid[w1.Join[x,{1}]],{1}])]
```

Assume the decision boundary is where the function crosses zero. Based on this assumption, classify the data based on these pseudo-decision boundaries. First, determine which of the training data points are classified correctly. This applies the regression function h to each of the data points and uses the sign of the results to determine the classification of each element.

```
calcres = Sign[h/@ T[Drop[T[Take[output,250]],-1]]];
```

The true classification is determined by looking at the sign of the target value from each vector.

```
actres = Sign /@ T[Take[output,250]][[4]];

concat = T[Join[T[Take[output,250]],{calcres,actres}]];
```

The data vectors classified as negative and actually negative are stored in the negs variable.

```
negs = Select[concat,(#[[5]]==-1 && #[[6]]==-1)&];
Length[negs]
100
```

Similary, the data vectors accurately classified as positive are stored in poss.

```
poss = Select[concat,(#[[5]]==1 && #[[6]]==1)&];
Length[poss]
145
```

Get rid of the extra stuff needed to determine if the data is classified correctly.

```
posr = T[Take[T[poss],3]];
negr = T[Take[T[negs],3]];
```

For each posr vector, find the nearest vector of negr.

```
nearToW0 = FindNearest[posr,negr];
```

For each negr vector, find the nearest posr vector.

```
nearToW1 = FindNearest[negr,posr];
```

Using the nearest neighbors of opposite sign determined above, use Mathematica's built in secant root finding method to find the decision boundary points.

```
db0 = intersect[#[[1]],#[[2]],h]& /@ T[{nearToW0,posr}];
db1 = intersect[#[[1]],#[[2]],h]& /@ T[{nearToW1,negr}];
```

Determine the gradient of the discriminant function at each of the decision boundary points.

```
norm = delY[#]& /@ Join[db0,db1];
```

Normalize each gradient to unit length.

```
norm1 = #/Sqrt[#.#]& /@ norm;
```

Build the effective decision boundary feature matrix.

```
sigEDBFM =
 Sum[
  Outer[Times,norm1[[i]],norm1[[i]]],
  {i,1,Length[norm1]}];
sigEDBFM//MatrixForm
0.0000118256   0.0226168      -0.048844
0.0226168      43.2556        -93.4161
-0.048844      -93.4161       201.744
```

Calculate the eigenvalues and eigenvectors:

```
{evals,evects} = Eigensystem[sigEDBFM/Length[norm1]];
evals
                -19
{1., 4.14499 10   , 0.}
T[evects]//TableForm
0.000219699   -0.0000319298   -1.
0.420182      0.90744         0.0000923136
-0.90744      0.420182        -0.000199363
```

Calculate saliencies:

```
sal = (T[evects]^2).evals
           -8
{4.82676 10  , 0.176553, 0.823447}
```

A.3 Mathematica Code

This section lists the code for calculation of the gradients and determination of nearest neighbors.

The derivative of the sigmoid function:

```
dsig[x_] := Evaluate[D[sigmoid[x],x]]
dsigo[x_] := Evaluate[D[sigmoido[x],x]]
```

The gradient of the output

```
delY[x_] :=
Module[
{zh},
zh = Join[sigmoid[w1.Join[x,{1}]],{1}];
T[
  Take[
    T[
      dsigo[ w2.zh]*w2.
      Join[
        dsig[w1.Join[x,{1}]]*w1,
  {Table[0,{j,Length[x]+1}]}]
],
```

```
    Length[x]
  ]
]]
```

The function which finds the decision boundary point:

```
intersect[pt1_,pt2_,f_] :=
Module[{x,u,v},
v = pt2 - pt1;
x[u_] := u v + pt1;
x[t] /. FindRoot[f[x[t]], {t,0,1}]
]
```

Find the closest member of the opposite class. This is done by installing a MathLink extension.

```
Install["find_nearest1"];
```

The template and code for find_nearest1

```
:Begin:
:Function:        find_nearest
:Pattern:         FindNearest[x_?MatrixQ,y_?MatrixQ]
:Arguments:       {x,y}
:ArgumentTypes:    {Manual}
:ReturnType:      Manual
:End:
#include "mathlink.h"
#include "stdlib.h"

void find_nearest(void) {
long   *x_dimensions;
char   **x_heads;
long   x_depth;
double *x_data;
```

```
long   *y_dimensions;
char   **y_heads;
long   y_depth;
double *y_data;
int    i, j, k;
double* best;
double dist;
double old_dist;
double t;
double* tptr;
double *tdata;

MLGetDoubleArray(stdlink, &x_data, &x_dimensions,
 &x_heads, &x_depth);
MLGetDoubleArray(stdlink, &y_data, &y_dimensions,
 &y_heads, &y_depth);

tdata =
  (double*)malloc(sizeof(double)*x_dimensions[0]*x_dimensions[1]);
if (!tdata) DebugStr("\pmemory allocation failed");
  /* go thru the entire list determining the closest y vector
     to each x vector. Store only a pointer to the y_vector here,
     will store in an array when done */
     for(i=0; i<x_dimensions[0]; i++) {/* go thru each x vector */
        old_dist = 100000000000.;
        for(k = 0; k<y_dimensions[0]; k++) {/*go thru each y vec */
           dist = 0;
           for(j=0; j<x_dimensions[1]; j++) {
```

```
  /* go thru each element */
            t = x_data[ j + i *  x_dimensions[1]] -
            y_data[ j + k *  x_dimensions[1]];
            dist += t*t;
          } /* j loop */
     if (dist < old_dist) {
       old_dist = dist;
       tptr = &(y_data[ k *  x_dimensions[1] ]);
     }  /* if */
       }  /* k loop */
  /* store the nearest y vector in the return array */
  for(j=0; j<x_dimensions[1]; j++)
       tdata[j + i *  x_dimensions[1]] = tptr[j];
     } /* i loop */

  /* return the array */
MLPutDoubleArray(stdlink, tdata, x_dimensions, x_heads, 2);

/* clean up */
free(tdata);
MLDisownDoubleArray(stdlink, x_data,
x_dimensions, x_heads, x_depth);
MLDisownDoubleArray(stdlink, y_data,
y_dimensions, y_heads, y_depth);
}

int main(argc, argv)
int argc; char* argv[];
```

```
{
return MLMain(argc, argv);
}
```

Appendix B. Annotated LNKmap Example

B.1 Introduction

This appendix provides an annotated example of the use of the Unix LNKmap function mapping software. This example is taken from the pairwise subset enumeration runs discussed in Chapter IV. All of the commands and functions used here are documented either in online *man* files or in the LNKnet User's Guide. The intent here is not to create a reference manual, but instead to provide an example and highlight the important details.

B.2 File Set Up

After preprocessing the data as discussed in Chapter III and creating the full data matrix, *Awk* can be used to extract the columns of interest for a particular run. For this example the RSI and 10 day stochastics subsets will be used to train a network. The *Awk* program file to extract the indicators from the full data matrix is:

```
{print $1,$8,$9,$10,$11,$16,$17,$18,$19,$20,$21}
```

This extracts the numbered columns from the master data matrix. Assuming this program is stored in the file r10.awk, the program can be executed on the master training and test data matrices with:

```
awk -fr10 master.train >sandp.train
awk -fr10 master.test > sandp.test
```

In addition LNKmap requires defaults files sandp.train.defaults and sandp.test.defaults. For this example the sandp.train.defaults file contains:

```
describe -ninputs 10 -noutputs 1 -npatterns 800 -map
```

This indicates that there are 800 patterns of 10 features with a single output and that the mapping mode is invoked. The test.defaults file differs only in the number of patterns, 300 instead of 800.

A directory is created for each run, and the data and defaults files are stored in this directory. The directory used for this example is *sandp10*.

B.3 Run Files

The executable file which begins training of the neural network is:

```
set loc='pwd'
(time mlpm \
-train  -create  -pathexp $loc  -ferror sandpmlpm.err.train \
 -fparam sandpmlpm.param\
 -pathdata /tmp_mnt/home/hawkeye6/95m/jastewar/LNKmap/sandp4 \
  -finput sandp.train\
 -fdescribe sandp.train.defaults  -nraw 10 \
 -npatterns 800 -fnorm sandp.norm.none\
 -cross_valid 0  -fcross_valid sandp.train.cv  -seed 1 \
 -debug 0  -verbose 3\
 -verror 0  -nodes 10,10,1 -alpha 0.8  -etta 0.01  \
 -epsilon 0.1  -kappa 0.01\
 -decay 0  -tolerance 0.001  -hfunction 0\
 -ofunction 1 -param 3.0 -epochs 12000\
 -batch 1,1,0  -init_mag 0.1 -random ) \
 |& nn_tee -h  sandpmlpm.log
echo "current directory:" >> sandpmlpm.log
echo $loc >> sandpmlpm.log
```

The main program is *mlpm*. The options are all described in the man pages for *classifier*, *mapper* or *mlp*. The `-nodes` option is the network architecture, in this case, the network will have 10 inputs, 10 hidden nodes and 1 output node. The parameters `-alpha` and `-etta` are the backpropagation momentum and stepsize respectively. The `-epochs` option determines when training stops. In this example, training will cease after 12000 passes through the training data have occurred. This run file is normally held in a script file and executed by typing the file name. When the training file is executed, it echoes the training root means square error for each epoch to the screen and a log file. The rmse can be monitored to ensure that learning is taking place.

It is much faster to modify the script file to try different step sizes or a different architecture than to use the graphic interface provided by LNKmap. Also, by creating multiple directories with data files and scripts set up for different runs, it is possible to use multiple workstations simultaneously to accomplish training and testing of different data sets. The graphic interface is useful for setting up the initial run and creating the script files, but after the files are created modifying the script files is more efficient than using the interface. In addition, modification of the script run files can be automated with other Unix tools.

When training is complete, the weights are frozen and a test run on both the training and the test set should be accomplished. The script file for testing is the same as the training script above, except the `-create` and `-train` options are omitted, as the network parameters are read from a parameter file. The `-finput` option must be changed to reflect either the training or the test set and `-verror` should be set to 1 to create a listing of the network outputs for each input. The rmse errors for each set are written to the screen and a log file.

B.4 Summary

This appendix has presented an overview of using the LNKmap software to do neural network based function mapping.

Bibliography

1. Atkinson, Kendall E. *An Introduction to Numerical Analysis* (second Edition). New York: John Wiley and Sons, 1989.
2. Azoff, E. Michael. *Neural Network Time Series Forecasting of Financial Markets*. New York: John Wiley and Sons, 1994.
3. Becker, John D. "Value of oscillators in determine price action," *Futures* (May 1994).
4. Brockwell, Peter J. and Richard A. Davis. *Time Series: Theory and Methods*. New York, New York: Springer-Verlag, 1991.
5. Casdagli, Martin. "Chaos and Deterministic versus Stochastic Non-linear Modelling," *Journal of the Royal Statistical Society B*, *54*(2):303–328 (1991).
6. Dillon, William R. and Matthew Goldstein. *Multivariate Analysis, Methods and Applications*. New York: John Wiley and Sons, 1984.
7. Etzkorn, Mark. "Getting an indication," *Futures* (February 1995).
8. Farmer, J. Doyne and John J. Sidorowich. "Predicting Chaotic Time Series," *Physical Review Letters*, *59*(8):845–848 (1987).
9. Foley, D.H. "Considerations of Sample and Feature Size," *IEEE Transactions on Information Theory*, *18*:618–626 (September 1972).
10. Garza, R. E., et al. "Neural Estimation and Embedology for Time Series Prediction." *Proceeeding of the 1995 SPIE Conference on Applications and Science of Artificial Neural Networks*. 1995. Submitted for publication.
11. Garza, Robert E. *Embedology and Neural Estimation for Time Series Prediction*. MS thesis, Air Force Institute of Technology, 1994.
12. Grassberger, Peter and Itamar Procaccia. "Measuring the Strangeness of Strange Attractors," *Physica D*, *9*:189–208 (1983).
13. Haesloop, Dan and Bradley R. Holt. "Neural Networks for Process Identification." *Proceedings of the International Joint Conference on Neural NetworksIII*. 429–434. 1990.
14. Kennel, Matthew B. and Henry D. I. Abarbanel, "False Neighbors and False Strands: A Reliable Minimum Embedding Dimension Algorithm." unpublished, anonymous ftp from Lyapunov.ucsd.edu.
15. Kennel, Matthew B. and Henry D. I. Abarbanel. "Local False Nearest Neighbors and Dynamical Dimensions from Observed Chaotic Data," *Physics Review E*, *47*:3057–3068 (1993).
16. Lee, Chulhee and David A. Landgrebe. "Decision Boundary Feature Extraction for Nonparametric Classification," *IEEE Transactions on Systems, Man and Cybernetics*, *23*(2):433–444 (1993).

17. Lee, Chulhee and David A. Landgrebe. "Feature Extraction Based on Decision Boundaries," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *15*(4):388–400 (1993).

18. Lee, Samuel E. and Bradley R. Holt. "Regression Analysis of Spectroscopic Process Data Using a Combined Architecture of Linear and Nonlinear Artificial Neural Networks." *Proceedings of the International Joint Conference on Neural NetworksIV*. 549–554. 1992.

19. Longinow, Nicholas E. "Predicting Pilot Look-Angle With a Radial Basis Function Network," *IEEE Transactions on Systems, Man and Cybernetics*, *24*(10):1511–1518 (1994).

20. Makridakis, Spyros, et al. *Forecasting: Methods and Applications* (second Edition). New York: John Wiley and Sons, 1983.

21. Pao, Yoh-Han. *Adaptive Pattern Recognition and Neural Networks*. Reading, Massachusetts: Addison-Wesley, 1989.

22. Priddy, K. L., et al. "Bayesian Selection of Important Features for Feedforward Neural Networks," *Neurocomputing* (1992).

23. Rogers, Steven K. and Mathew Kabrisky. *An Introduction to Biological and Artificial Neural Networks for Pattern Recognition*. Bellingham, Washington: SPIE Optical Engineering Press, 1991.

24. Ruck, Dennis, et al. "Feature Selection Using a Multilayer Perceptron," *Journal of Neural Network Computing*, *2*(2):40–48 (1990).

25. Rumelhart, D.E., et al. "Learning Internal Representations by Error Propagation." *Parallel Distributed Processing: Explorations in the Microstructures of Cognition 1*, edited by D.E. Rumelhart and J.L. McClelland, MIT Press, 1986.

26. Sauer, Tim. "Time Series Prediction by Using Delay Coordinate Embedding." *Time Series Prediction: Forecasting the Future and Understanding the Past* edited by Andreas S. Weigend and Neil A. Gershenfeld, 175–193, Addison-Wesley, 1994.

27. Steppe, Jean M. *Feature and Model Selection in Feedforward Neural Networks*. PhD dissertation, Air Force Institute of Technology, 1994.

28. Stright, James R. *A Neural Network Implementation of Chaotic Time Series Prediction*. MS thesis, Air Force Institute of Technology, 1988.

29. Stright, James R. *Embedded Chaotic Time Series: Applications in Prediction and Spatio-Temporal Classification*. PhD dissertation, Air Force Institute of Technology, 1994.

30. Takens, Floris. "Detecting strange attractors in turbulence." *Dynamical Systems and Turbulence, Warwick 1980*, Lecture Notes in Mathematics 898, edited by D. A. Rand and L. S. Young. 366–381. Springer-Verlag, 1981. Printed in Germany.

31. Tarr, G. L. *Multi-Layered Feedforward Neural Networks for Image Segmentation.* PhD dissertation, Air Force Institute of Technology, 1991.

32. Theiler, James. "Estimating fractal dimension," *Journal of the Optical Society of America A*, *7*(6):1055–1073 (1990).

33. Theiler, James, et al. "Testing for nonlinearity in time series: the method of surrogate data," *Physica D*, *58*:77–94 (1992).

34. Waibel, Alexander, et al. "Phoneme Recognition Using Time-Delay Neural Networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *37*(3):328–339 (1989).

35. Wan, Eric A. "Time Series Prediction by Using a Connectionist Network with Internal Delay Lines." *Time Series Prediction: Forecasting the Future and Understanding the Past* edited by Andreas S. Weigend and Neil A. Gershenfeld, 195–218, Addison-Wesley, 1994.

36. Weigend, Andreas S. and Neil A. Gershenfeld. *Time Series Prediction: Forecasting the Future and Understanding the Past.* Reading, Massachusetts: Addison-Wesley, 1994.

37. Yule, G. "On a Method of Investigating Periodicity in Disturbed Series with Special Reference to Wolfer's Sunspot Numbers," *Phil. Trans. Roy. Soc.*, *A*(226):267–298 (1927).

Vita

Capt James A. Stewart was born at Laughlin AFB, Texas on 22 November, 1962. He graduated from high school in Hondo, Texas in 1981, and received a bachelors degree in Aerospace Engineering from the University of Texas at Austin in 1985. He was commissioned through the Air Force Reserve Officer Training Corps at the University of Texas. Upon graduation he made a short visit back to Laughlin AFB, in an unsuccessful attempt to become a pilot. He then moved to Mather AFB, California to attend Undergraduate Navigation Training, which he completed in December 1986. After a three month visit to Castle AFB, California for KC-135 Combat Crew Training School, he was assigned to Plattsburgh AFB, New York from 1987 until 1993 as a KC-135 navigator. He entered the Masters program in the School of Engineering, Air Force Institute of Technology in August, 1993.

James married the former Janis R. Lents in 1983, and they have four children, Amanda, Tristan, Elizabeth, and Trevor.

Permanent address: HCR 2, Box 67
Yancey, TX 78886